INTEGER PROGRAMMING BASED SEARCH

A Thesis
Presented to
The Academic Faculty

by

Michael R Hewitt

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Industrial and Systems Engineering

Georgia Institute of Technology
December 2009

INTEGER PROGRAMMING BASED SEARCH

Approved by:

Professor Martin Savelsbergh, Advisor
School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor Alan Erera, Co-Advisor
School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor George Nemhauser,Co-Advisor
School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor Ozlem Ergun
School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor Mark Ferguson
College of Management
Georgia Institute of Technology

Date Approved: August 2009

ACKNOWLEDGEMENTS

I would like to thank my advisors Alan Erera, George Nemhauser and Martin Savelsbergh for the unique perspective and expertise they each brought to this thesis and my graduate studies. I valued both greatly. I could not have asked for more from doing research with the three of them; it simply has been a lot of fun. I would especially like to thank Martin Savelsbergh for his involvement and patience with my maturation as a researcher and writer.

I would like to thank the industry partners Yellow Roadway Corporation, Saia. Inc and Exxon Mobil Corporation, who have sponsored my research at one point or another. Each of them reminded and reassured me of the practical importance of the work I was doing.

Finally, I'd like to thank my wife and daughter for letting me work so much but not all the time. Don't worry. Tenure is only 6 years away!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

When integer programming (IP) models are used in operational situations there is a need to consider the tradeoff between the conflicting goals of solution quality and solution time, since for many problems solving realistic-size instances to a tight tolerance is still beyond the capability of state-of-the-art solvers. However, by appropriately defining small instances, good primal solutions frequently can be found quickly. We explore this approach in this thesis by studying the design of algorithms that produce solutions to an integer program by solving restrictions of the problem via integer programming technology. We refer to this type of algorithm as IP-based search.

This approach is also taken, for example, within LP-based branch-and-bound algorithms using techniques such as Local Branching and Relaxation Induced Neighborhood Search (RINS). These techniques use information from the LP solution and incumbent solution to define a small IP, which is then optimized. These techniques can be applied to any integer program and are available in commercial solvers such as CPLEX. We develop new IP-based search approaches for specific problems that exploit problem structure and an approach that can be easily applied to general integer programs. Finally, we leverage some of the strengths of IP-based search to develop new and more accurate models of a network design problem faced by freight transportation carriers.

In the first part of the thesis we present a heuristic for the classical Multi-Commodity Fixed Charge Network Flow (MCFCNF) model that exploits problem structure to produce high quality solutions quickly. The solution approach combines mathematical programming and heuristic search techniques. To obtain high-quality solutions it relies on neighborhood search with neighborhoods that involve solving carefully chosen integer programs derived from the arc-based formulation of MCFCNF. To obtain lower bounds, the linear programming relaxation of the path-based formulation is used and strengthened with cuts discovered during the neighborhood search. Computational experiments demonstrate that

the proposed approach outperforms both best-known meta-heuristics and a state-of-the-art MIP solver.

In the second part of the thesis we present an IP-based search algorithm for mixed integer programs that does not depend on problem structure. We formalize IP-based search as solving a restriction of the original problem and then develop an extended formulation to model the choice of restriction to solve. We propose a parallelized branch-and-price scheme for solving the extended formulation that is designed to produce high quality solutions quickly. We illustrate the application of the algorithm on the MCFCNF and computational experiments indicate it is competitive both with a state-of-the-art MIP solver and the structure-based heuristic presented in the previous part, even though it is a more general algorithm.

Lastly, the thesis addresses the applicability of IP-based search to a real-world problem; namely the Service Network Design problem faced by Less-Than-Truckload (LTL) freight transportation carriers. In this part we present advances both in modeling and algorithm design. The developed models more accurately capture key operations of today's carriers: decisions for loaded and empty trailer movements are considered simultaneously, and a time discretization is used that can appropriately model the timing of freight consolidation opportunities. Along with providing decision support for traditional service network plans used by LTL carriers, the models also enable the development of plans that allow more flexibility, such as allowing certain freight routes to vary by weekday. Given the additional detail within the proposed models, very large problem instances result when they are applied to large-scale LTL networks. Yet computational experiments using data from a large U.S. carrier demonstrate that the proposed modeling and IP-based search approach has the potential to generate significant cost savings.

CHAPTER I

INTRODUCTION

For many optimization problems, no algorithms guaranteed to find an optimal solution for large instances in a reasonable amount of time are known or are likely to be found. Yet these problems are of great practical interest. For example, the traveling salesman problem can be used to determine both the route of a small package delivery driver and the order in which holes should be drilled in a circuit board. Models from the class of network design problems, the primary focus of this thesis, capture the keys decisions that many transportation companies must make in order to serve clients at low cost.

There are many different ways to search for an optimal or high-quality solution to a difficult discrete optimization problem. Branch-and-bound [24] recursively partitions the set of feasible solutions into regions, searching one exhaustively when it is deemed "easy" to do so. Typically, these regions are modeled as *restrictions* of the original problem and are simple in nature, such as fixing the value of a binary variable to 0 or 1. Many techniques have been developed to select the next region to partition, and bounding arguments are used to determine those that cannot contain an optimal solution and thus need not be searched. Although much work has been done to speed up branch-and-bound-based algorithms, particularly in the area of generating strong bounds during branch-and-cut, it is still an exponential time algorithm.

With the global nature of today's economy and an increased emphasis on time-definite transportation services, real-life network design instances are very large and thus nearly impossible to solve to optimality. While the strategy employed by branch-and-bound guarantees that an optimal solution will be found, there is no guarantee and little likelihood of finding a high quality solution quickly.

Neighborhood or local search heuristics [1] present an alternative approach to searching for solutions. These techniques sacrifice the guarantee of finding an optimal solution to find

high quality solutions quickly. Instead of producing solutions by searching systematically-defined regions of the feasible set, these heuristics iteratively define and search a neighborhood of a known solution for an improving solution. Thus, critical decisions when designing a local search heuristic are the size of the neighborhood to be searched at an iteration and whether its structure enables the neighborhood to be searched efficiently. Should it be small and "easy" to search? If so, the heuristic can perform many searches in a fixed period of time but risks becoming stuck at a bad local optimum. To avoid this phenomenon, many have studied the design of meta-heuristics [6], which are high-level strategies used to guide a more problem-specific search heuristic. By structuring the sequence of small neighborhoods searched, meta-heuristics have been quite successful at avoiding becoming stuck at bad local optima. Although they are often successful at producing high-quality solutions, local search heuristics are typically unable to produce a bound on the optimal value of the problem. Thus the only measure of solution quality is a comparison with solutions produced by alternative approaches.

In this thesis we study the design of algorithms that produce solutions to an integer program (IP) by solving restrictions of the problem via integer programming technology. We refer to this type of algorithm as IP-based search. In addition to the simple variable fixing restriction structure used in branch-and-bound, we also consider more general structures. To guide our search we use different mechanisms both for creating restrictions and choosing the one we solve next. We consider modeling a problem with a formulation that captures both the problem and the choice of restriction to solve to produce its optimal solution. Then, by searching for solutions to this extended formulation with a branch-and-bound-based algorithm, we ensure that we can produce dual bounds and have an exact algorithm. In the spirit of local search, we also consider building restrictions that represent a neighborhood of a known solution. In contrast to meta-heuristics, we are trying to avoid getting stuck at a bad local optimum by trading a large number of searches for neighborhoods that are large enough to contain a locally optimal solution of high-quality. While we use neighborhoods similar (exponential) in size to Very Large Neighborhood Search [2], by not confining ourselves to those that can be searched efficiently, they are *theoretically* much harder to search.

However, by coupling intelligent neighborhood selection mechanisms with the power of today's commercial IP solvers, *in practice* these neighborhoods can be searched quickly. The idea of modeling the neighborhood of a solution as the set of feasible solutions to an integer program and then searching that neighborhood with integer programming technology has recently garnered a great deal of attention in the literature. For general integer programs, both Relaxation Induced Neighborhood Search (RINS, [12]) which creates variable fixing restrictions by combining information from a feasible solution and a solution to a linear program (LP), and Local Branching [14] which creates restrictions that restrict the number of binary variables whose value may differ from their value in a known solution have been very successful. For a structured problem, the heuristic presented in [13] for the Distance-Constrained Capacitated Vehicle Routing Problem is interesting both for its success and the fact that it solves integer programs that are not restrictions of the problem being solved.

Local search and exact optimization approaches often differ both in their emphasis and the techniques they use. While exact optimization algorithms produce dual bounds and proofs of optimality, they often do so at the expense of finding high quality solutions early in the search process. On the other end of the spectrum, local search heuristics often produce good solutions quickly but no dual bound. However, dual bounds, proofs of optimality and finding high quality solutions quickly are all desirable algorithmic properties. In addition, it is natural when designing an algorithm to consider what techniques can be borrowed from approaches developed by both the heuristic and mathematical programming communities. One goal of this research is to study the design of IP-based search algorithms that combine ideas from these communities both to guide the search for good solutions and produce dual bounds. In studying IP-based search we first consider two questions:

1. **Can we design IP-based search algorithms that quickly produce good solutions?** On problems of both academic and real-world interest we show that we can, and that the solutions produced are often near-optimal. In addition, the solutions produced after a short period of time by these IP-based search algorithms are often much better than those produced by both state-of-the art commercial optimization solvers and meta-heuristics, even when these other algorithms are given significantly

more time. We also see that IP-based search algorithms are particularly effective for extremely large problem instances. While branch-and-bound begins its search at the root node of a tree with no extra restrictions placed on the variables of an IP, one can think of IP-based search as pruning nodes at an intermediate depth wherein many variables have their value fixed. These fixed variables obviate the need to load a full instance into memory, which in itself can cause problems. In addition, by fixing variables cleverly, we may be left with restrictions of the IP that have structure that can be exploited. We also see that by defining the search procedure as solving an integer program, IP-based search algorithms are flexible since considering different problems from a specific class only requires redefining the integer program.

2. **Can we design an IP-based search algorithm which provides a measure of the quality of the solution produced?** We focus on problems that have both a compact and extended formulation and actively use the two. A well-known fact from column generation is that one can couple the solution of the linear programming (LP) relaxation of an extended formulation with lagrangean techniques to generate a dual bound that is no worse and often stronger than the dual bound provided by the compact formulation. In addition, this fact provides a mechanism for producing a dual bound for problems whose size prohibits solving the LP relaxation of the compact formulation.

We address these questions in the context of network design problems by developing IP-based search algorithms for the Multi-commodity Fixed Charge Network Flow (MCFCNF) problem and the load plan design problem encountered in the Less-Than-Truckload (LTL) freight transportation industry. While MCFCNF allows us to study the effectiveness of IP-based search in a more academic setting, the latter problem allows us to study its applicability to a real-life transportation problem that includes complicating side constraints.

MCFCNF is a classic optimization problem that lies at the heart of many transportation problems. In MCFCNF, a set of commodities must be routed through a directed network, each of which has an origin and a destination. The MCFCNF models both the routing of

commodities and the design of the network itself by charging a fixed cost when an arc is used. While there are many variants of MCFCNF, we focus on instances with capacitated arcs, but consider both the case where the demand of a single commodity must follow a single path and the case where it may be split across multiple paths. Regarding cost structure, we associate a fixed cost with using an arc and a variable cost that depends on the quantity routed along the arc. The objective is to minimize the total cost. We first present an IP-based neighborhood search heuristic that solves restrictions derived from the arc-based formulation of MCFCNF wherein a subset of variables is fixed to their value in the best known solution. Like Variable Neighborhood Search [23], the heuristic first chooses a class of variables to fix and then uses ideas ranging from studying the structure of good solutions found so far to deriving information from a solution to the LP relaxation of the extended (path-based) formulation of MCFCNF to determine which variables of that class to fix.

One mechanism for assessing the quality of an integer MCFCNF solution is to combine the LP relaxation of the path-based formulation with lagrangean techniques to produce a dual bound on the optimal IP value. However, for MCFCNF, the LP relaxation alone is known to produce a weak dual bound and valid inequalities are critical to producing a meaningful lower bound. Another innovative aspect of the heuristic is that the LP relaxation is strengthened by cuts which are found during the solution of the small IPs.

Computational results indicate that IP-based search can quickly produce high-quality solutions to the MCFCNF. For instances of various sizes, the quality of the solution produced by IP-based search in only 15 minutes is often significantly better than the best solution found by the state-of-the-art commercial solver CPLEX or a best-known meta-heuristic, even when those approaches are given much more time.

Next, after observing the success of an IP-based search heuristic that is designed for a specific problem class and uses a specific type of restriction, we present an approach that generalizes some of the ideas and techniques used. In particular, we do not require that restrictions are formed by fixing variables, but instead by a set of constraints added to the original problem. We also present methods for creating and selecting restrictions that are

systematic but not problem-specific; thus the resulting approach is a general integer programming algorithm. One of the novel aspects of the approach is that it creates restrictions by dynamically adding variables to the problem formulation via column generation. While the heuristic presented in Chapter 2 uses information from an extended formulation as one of many guides in the process of creating IPs to solve in its search for primal solutions, here we explicitly formalize the creation of restrictions by solving an optimization (pricing) problem. Given that we have a potentially huge number of restrictions, we are still left with the questions of which one we should solve next and whether we can guarantee that we find the optimal solution without solving them all. Fortunately, branch-and-price [5] gives us an effective and well-understood framework for making these decisions since it is specifically designed to solve integer programs that are modeled with an *extended* formulation that has more variables than can be considered explicitly.

Although our algorithm falls into the category of a branch-and-price approach, it differs significantly from those in the literature. An extended formulation is typically chosen because it provides a stronger linear relaxation than the usual (*compact*) formulation for a problem. Therefore, an extended formulation is typically used to strengthen the dual bound produced throughout the course of a linear programming based branch-and-bound method. Our extended formulation is designed to facilitate creating restrictions of the compact formulation of the problem that are small enough to be solved quickly. Thus, the extended formulation improves our ability to produce primal solutions. In addition, an extended formulation is typically structurally different from the compact formulation since it involves modeling a decision with different "objects" (i.e. choosing a path for routing a commodity as opposed to choosing a set of arcs). As a result, effective valid inequalities for the compact formulation may not be applicable to the extended formulation. By simply adding variables to the compact formulation, our extended formulation allows the use of all inequalities that are valid for the compact formulation.

While branch-and-price schemes traditionally only use LP bounds to prune nodes in the search tree, recent procedures such as [26] employ the idea that when deep enough in the branch-and-bound tree wherein many variables are fixed, it is best to switch back to

the compact formulation of the problem and let the IP solver prune the node. In addition to improving the dual bound by pruning a node, switching to the compact formulation and solving the IP also aids in the search for good primal solutions. With our extended formulation, we can easily create and solve a restriction of the compact formulation to produce a feasible IP solution at every node of the branch-and-bound tree such that one exists. We also show that our extended formulation provides easy and general ways to create integer programs that represent neighborhoods of the best known solution.

To illustrate the application of the approach, we return to the single-path variant of the MCFCNF. The computational experiments show that the approach can quickly produce high quality primal solutions and tight dual bounds. For instances of various sizes, the approach often produces in 15 minutes a primal solution which is both near-optimal and better than what CPLEX can produce in 6 hours and a dual bound that is comparable to what CPLEX can produce in 6 hours. In addition, the dual bound produced by our approach in 15 minutes often provides a *proof* that our solution is optimal (or near-optimal).

Lastly, we address the effectiveness of IP-based search on a real-life network design application, the load plan design problem in the Less-Than-Truckload (LTL) freight transportation industry. In addition to presenting a new IP-based search algorithm for producing cost-effective load plans, we develop new models to accurately capture how carrier's currently operate and to study the cost effectiveness of new business practices. In particular, we develop a model of LTL operations that differs significantly from existing load plan design research in three ways; it models time at a level that is appropriate to tight service standards, captures the interplay between empty and loaded trailer routing decisions and can support freight routing decisions that vary by day. The first two of these advances improve our ability to design cost-effective load plans that carriers find implementable, while the last allows us to study a new mode of operation for LTL carriers that has significant potential for cost savings.

The transportation industry is one of the largest in the U.S. with \$1.4 trillion (10.6% of total GDP) spent on transportation-related goods and services in 2006. In addition, the transportation industry represents 10% of the total U.S. labor force. The trucking industry

now accounts for 6% of GDP. The trucking industry has two sectors: Full Truckload (FTL or TL) and Less-Than-Truckload (LTL) with TL accounting for the major share of revenues (about 70% of the market is TL).

While a truckload carrier transports freight directly from origin to destination without the freight being handled en route, the size of LTL shipments (typically between 100 and 10,000 lbs) render it too costly to transport each customers' shipment directly from origin to destination. As a result, LTL carriers collect freight from various shippers and consolidate that freight in order to build nearly full trailers.

The LTL industry has changed significantly over the last 20 years with many regional carriers consolidating into *super-regional* or national carriers. In addition, UPS and FedEx have entered the marketplace, increasing competition in both price and service. While historically a shipment was quoted a service standard from origin to destination of 5 business days, today's LTL customers often expect service standards of 1,2 and 3 days. To accommodate increased competition on price, LTL carriers must find new ways to increase the utilization of their current network infrastructure in spite of these tighter service constraints.

The utilization of a carrier's network is driven by the *load plan* which determines shipment consolidation opportunities by prescribing how freight is routed from origin to destination and where it is handled. The concept unique to load plan design is that of a *direct*, which specifies where freight is handled. For example, saying a shipment takes the direct $Atlanta \rightarrow Detroit$ means the shipment is loaded into a trailer in the Atlanta terminal which is not opened (and hence the freight not handled) again until it reaches the Detroit terminal. The local search heuristic presented by [28] (the engine within the commercial software used by nearly every LTL carrier at one point in time) leverages this property in its search for a better load plan by performing a sequence of add and drop direct operations and re-routing freight flows after each operation. However, the heuristic does not explicitly model daily freight volume fluctuations, service standards or the timing of consolidation opportunities and does not route loaded and empty trailers simultaneously.

The path a shipment follows in an LTL network consists of a sequence of directs. The load plan prescribes how a shipment is routed through the carrier's network of directs by

specifying the unique direct to take given the shipment's current terminal location and ultimate destination terminal. For example, the load plan may give the following instruction: "all freight in Jackson, TN destined for Atlanta, GA loads direct to Nashville, TN." Thus, the load plan dictates that freight follows a unique path within the network of directs and that the directs into a destination terminal must form a directed in-tree. We present an IP-based search heuristic that is motivated by this structural property in that it searches for a better load plan by solving an integer program to optimize the routing of all freight destined for a given terminal while holding fixed freight destined for all other terminals. While the work presented in [21] also bases the search for load plans on this in-tree structure, it neither models time as precisely as we do nor does it route loaded and empty trailers simultaneously. In addition, we present valid inequalities and preprocessing techniques to speed up the solution of these integer programs and heuristics to control their size without forgoing consolidation opportunities.

Computational results indicate that IP-based search is very effective for the load plan design problem. To test the heuristic designed we use the existing load plan of a national carrier as a baseline and historical freight volumes as a data set. Although our detailed model of time yields networks with nearly 6,000 nodes and 500,000 arcs and full problem instances with over 1,000,000 integer variables and 2,000,000 constraints, we see that IP-based search can still produce load plans with proposed weekly savings of 3% in transportation and handling costs.

To respond to the challenges presented by increased competition in both price and service, carriers are interested not only in better load plans but also in new modes of operation. One new mode of operation, heretofore not considered in the literature, is the use of predictable daily freight volume variations in the load plan design process to build plans that vary by weekday. The heuristic we present can easily accommodate this variation of the traditional load plan and computational results indicate that by allowing routing decisions to vary by day we can save nearly 6% in weekly transportation and handling costs.

This thesis makes contributions both to algorithm design by illustrating the effectiveness

of integer programming based search and the study of network design for freight transportation carriers by developing new models of LTL operations.

Our main algorithmic contributions show that:

- IP-based search algorithms can quickly produce high-quality solutions for network design problems of both theoretical and real-world interest;
- IP-based search algorithms are particularly well-suited to large problem instances and classes of problems that contain many variants;
- we can use an extended formulation of a problem both to guide the search for good primal solutions and generate dual bounds; and
- we can design an exact IP-based search algorithm that will converge to the optimal solution without sacrificing solution quality in the early stages of the algorithm.

The main contributions to the study of network design for freight transportation carriers fall into two categories; studying the potential of new modes of operation and developing models which more accurately capture operational-level concerns. In particular, for the load plan design problem we show that:

- significant savings can be achieved by allowing freight routing decisions to vary by day;
- one can model time at the level of detail required in the presence of tight service standards without sacrificing the quality of load plan produced; and
- by integrating loaded and empty trailer routing decisions we can produce load plans which are both more cost effective and realistic than current practice.

CHAPTER II

IP-BASED NEIGHBORHOOD SEARCH

We begin our study by developing an IP-based local search heuristic for a specific problem (or class of problem); namely the Multi-commodity Fixed Charge Network Flow (MCFCNF) problem. The MCFCNF problem is a classic discrete optimization problem in which a set of commodities has to be routed through a directed network. Each commodity has an origin, a destination, and a quantity. Each network arc has a capacity. There is a fixed cost associated with using an arc and a variable cost that depends on the quantity routed along the arc. The objective is to minimize the total cost. Two versions of the problem are considered: commodities have to be routed along a single path and commodities can be routed along multiple paths. Many real-life instances of MCFCNF (or of instances of models that contain MCFCNF as a substructure) are very large (see for example [33]) - so much so that sometimes even the linear programming relaxation of the natural arc-based integer programming formulation can be intractable. In such situations, we can use a path-based integer programming formulation, but that necessitates the use of column generation techniques. Furthermore, the linear programming relaxation of the arc-formulation and the path-formulation have the same optimal value which is known to be weak. Hence, it may not just be difficult to find feasible solutions, it may also be challenging to determine the quality of these solutions. As a result, even though much research has been devoted to MCFCNF, exact methods are only capable of handling small instances, far smaller than many realistic-sized instance. Fortunately, in today's dynamic business environment getting high-quality solutions in a short amount of time is usually more important than getting provably optimal solutions. Hence the focus of this chapter is to develop a solution approach for MCFCNF that produces *provably* high-quality solutions *quickly.*

Our solution approach relies heavily on linear and integer programming to take advantage of the power of commercially available linear and integer programming solvers.

Algorithmically, we combine mathematical programming techniques with heuristic search techniques. More specifically, we develop

- a primal local search algorithm that utilizes neighborhoods that are not searchable in polynomial-time and instead are searched by an integer programming solver,
- a scheme to generate dual bounds that involves strengthening the linear programming relaxation via cuts discovered while solving these integer programs.

Our approach tightly integrates the use of the arc-based formulation of MCFCNF and the path-based formulation of MCFCNF. It also incorporates randomization to diversify the search and learning to intensify the search.

The resulting solution approach is very effective. For instances with 500 nodes, with 2000, 2500 and 3000 arcs, and with 50, 100, 150, and 200 commodities, we compared the quality of the solution produced by our solution approach with the best solution found by CPLEX after 15 minutes of computation and after 12 hours of computation. On average, the solution we found in less than 15 minutes is 35% better than CPLEX best solution after 15 minutes and 20% better than CPLEX best solution after 12 hours. Furthermore, we find a better solution than CPLEX best solution after 15 minutes within 1 minute, and CPLEX' best solution after 12 hours within 3 minutes. On these instances the approach produces dual bounds that are 25% stronger than the LP relaxation. We also compared the quality of the solutions produced by our solution approach with the quality of the solutions produced by a recent implementation of the tabu search algorithm of [15]. For nearly all instances in their test set, our solution is better than the solution of the tabu search algorithm and this solution is found much faster.

The key characteristics of our solution approach for MCFCNF, which differentiate it from existing heuristic approaches, are:

- it uses exact methods to find improving solutions,
- it generates both a primal solution and a dual bound at each iteration, and
- it uses both the arc and path formulations of MCFCNF to guide the search.

2.1 Literature

Metaheuristics have been developed that find good primal solutions to instances of MCFCNF. A tabu search algorithm using pivot-like moves in the space of path-flow variables is proposed in [10]. The scheme has been parallelized in [8]. A tabu search algorithm using cycles that allow the re-routing of multiple commodities is given in [15]. This cycle-based neighborhood is incorporated within a path-relinking algorithm in [16].

Our heuristic solves carefully chosen integer programs to improve an existing solution. As such, it considers exponential-sized neighborhoods similar to very large-scale neighborhood search (VLSN, [2]). However, in contrast to VLSN, no polynomial-time algorithm exists for searching these neighborhoods. General integer programming heuristics such as local branching [14] and relaxation-induced neighborhood search [12] also are integer-programming based local search algorithms, but they are different from ours.

The dual side of the problem is studied in [7], using various lagrangean relaxations and solution methods. Although lagrangean-based heuristics, such as the one proposed in [19] generate dual bounds, these bounds are no better than the value of the LP Relaxation. To the best of our knowledge, no heuristic produces dual bounds that are stronger than the LP relaxation.

Combining exact and heuristic search techniques, a key characteristic of our approach, has received quite a bit of attention in recent years, see for example [13], [36], [3], and [35].

2.2 Formulations

Before describing the main ideas of the proposed solution approach, we present the arc-based formulation and the path-based formulation for MCFCNF. Let $D = (N, A)$ be a network with node set N and directed arc set A. Let K denote the set of commodities, each of which has a single source $s(k)$, a single sink $t(k)$, and a quantity d^k that must be routed from source to sink. Let f_{ij} denote the fixed cost for using arc (i, j), c_{ij} denote the variable cost for routing one unit of flow along arc (i, j) and u_{ij} denote the capacity of arc (i, j). We use variables x_{ij}^k to indicate the fraction of commodity k routed along arc (i, j) and binary variables y_{ij} to indicate whether arc (i, j) is used or not. The arc-based formulation

of MCFCNF (Arc-MCFCNF) is:

$$\min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij}(d^k x_{ij}^k) + \sum_{(i,j) \in A} f_{ij} y_{ij}$$

subject to

$$\sum_{j:(i,j) \in A} x_{ij}^k - \sum_{j:(j,i) \in A} x_{j,i}^k = \delta_i^k \quad \forall i \in N,\ \forall k \in K, \tag{1}$$

$$\sum_{k \in K} d^k x_{ij}^k \leq u_{ij} y_{ij} \ \ \forall (i,j) \in A, \tag{2}$$

$$y_{ij} \in \{0,1\} \ \ \forall (i,j) \in A. \tag{3}$$

If a commodity's demand must follow a single path we have

$$x_{ij}^k \in \{0,1\} \quad \forall k \in K,\ \forall (i,j) \in A, \tag{4}$$

otherwise, we have

$$0 \leq x_{ij}^k \leq 1 \quad \forall k \in K,\ \forall (i,j) \in A. \tag{5}$$

The objective is to minimize the sum of fixed and variable costs. Constraints (1) ensure flow balance, where δ_i^k indicates whether node i is a source ($\delta_i^k = 1$), a sink ($\delta_i^k = -1$) or an intermediate node ($\delta_i^k = 0$) for commodity k. Constraints (2) are the coupling constraints that ensure that an arc is used if and only if its fixed charge is paid and that the total flow on the arc does not exceed its capacity. It is well-known that Arc-MCFCNF has a weak LP relaxation and can be strengthened by disaggregating the coupling constraints to

$$x_{ij}^k \leq y_{ij} \quad \forall k \in K,\ \forall (i,j) \in A. \tag{6}$$

The resulting formulation has a tighter LP relaxation, but comes at the expense of many more constraints.

Next, we consider the path-based formulation of MCFCNF. Let variable x_p^k denote the fraction of commodity k that uses path p and let $P(k)$ denote the set of feasible paths for commodity k. We will consider instances where $P(k)$ is not known in full. Hence let $\bar{P}(k) \subseteq P(k)$ denote a subset of all feasible paths for commodity k. The path-based formulation of MCFCNF (Path-MCFCNF) is:

$$\min \sum_{k \in K} \sum_{p \in P(k)} \Big(\sum_{(i,j) \in p} c_{ij} \Big) d^k x_p^k + \sum_{(i,j) \in A} f_{ij} y_{ij}$$

subject to

$$\sum_{p \in P(k)} x_p^k = 1 \quad \forall k \in K, \tag{7}$$

$$\sum_{k \in K} \sum_{p \in P(k):(i,j) \in p} d^k x_p^k \leq u_{ij} y_{ij} \ \ \forall (i,j) \in A, \tag{8}$$

$$y_{ij} \in \{0,1\} \ \ \forall (i,j) \in A. \tag{9}$$

We again have either $x_p^k \in \{0,1\}$ or $0 \leq x_p^k \leq 1$ depending on whether a commodity's demand may be split across multiple paths.

As before, the objective is to minimize the sum of fixed and variable costs. Constraints (7) ensure that every commodity is routed through the network and the coupling constraints (8) ensure that fixed charges are paid and that arc capacities are respected. Here too we can disaggregate the coupling constraints:

$$\sum_{p \in P(k):(i,j) \in p} x_p^k \leq y_{ij} \ \ \forall k \in K, \ \ \forall (i,j) \in A. \tag{10}$$

2.3 Solution Approach

At the heart of the primal side of our solution approach is a neighborhood search procedure. Consider the arc-based formulation of MCFCNF. A subset of variables V can be used to define a neighborhood of the current solution by fixing the values of variables $v \notin V$ to their value in the current solution. By selecting a suitably small subset of the variables a tractable integer program can be defined and solved using an IP solver. Hopefully, a better solution to the whole problem is obtained. The process can be repeated multiple times by choosing different subsets of variables. A pseudo-code describing this process is given in Algorithm 1.

Algorithm 1 Neighborhood Search

```
while the search time has not exceeded a prespecified limit T do
  Choose a subset of variables V
  Solve the IP defined by variables in V
  if an improved solution is found then
    Update the global solution
  end if
end while
```

Note that the neighborhood in Algorithm 1 is searched using an integer programming solver. The key to making this neighborhood search scheme work is in the choice of the subsets of variables V. This approach can be used on extremely large instances because the algorithm never requires the full instance to be in memory.

To evaluate the quality of the solution produced by the neighborhood search we find lower bounds on the value of the primal solution using a path-based formulation. While the number of variables in the path-based formulation is huge, variables can be considered implicitly rather than explicitly. The LP relaxation is solved over a subset of the variables and a pricing problem is solved to determine whether there is a need to expand the set of variables or not. The pricing problem is a shortest path problem and relies on the dual values associated with the constraints of the path-based formulation. A well-known observation related to this column generation process is that a valid lower bound can be obtained at every pricing iteration. Specifically, let z^*_{LP} be the value of the solution to the LP and let $\bar{c}^k_p$ be the reduced cost of an optimal path p for the pricing problem for commodity k. Then $z^*_{LP} + \sum_{k \in K} \bar{c}^k_p$ is a lower bound for the value of the LP when all columns are considered.

Note that any solution of Arc-MCFCNF can be converted to a solution of Path-MCFCNF and vice versa. As a result, a solution to the IP based on a subset of arc variables V can be converted to a set of variables for Path-MCFCNF. It is highly likely that several of these variables do not appear in the path formulation yet. In this way, the neighborhood search generates variables for Path-MCFCNF. Similarly, a solution to Path-MCFCNF can be converted to a solution of Arc-MCFCNF and can thus be used to guide the choice of subset V in the neighborhood search.

2.3.1 Neighborhood Search

During the course of the algorithm, we solve smaller integer programs defined by a subset of arcs in the network or a subset of commodities in the instance. We define a $SOLVER$ to be one of the two small integer programs together with a selection method for the necessary subset. A high-level implementation of a solver is given in Algorithm 2. A high-level implementation of the approach is given in Algorithm 3. We currently use a stopping

Algorithm 2 Solver Template

Require: A feasible solution F to the full MCFCNF instance
Require: The set of all feasible paths P found so far
  Select Subset
  Solve Subproblem
  Construct Feasible Solution $F^{'}$ to full MCFCNF instance
  Determine new paths $P^{'}$ found
  Determine new cuts $C^{'}$ found in course of solving subproblem
  return $(F^{'}, P^{'}, C^{'})$

criterion of whether the search time has exceeded a prespecified limit T. Given that we have

Algorithm 3 Neighborhood Search

  Find an initial feasible solution
  Set best_solution = initial feasible solution
  Set $PATHS$ = paths from initial feasible solution
  Set $SOLVERS$ = set of solvers we wish to consider using
  Set $CUTS = \emptyset$
  while not done **do**
    Solve path formulation LP only considering $PATHS$ but with $CUTS$ and let z^*_{LP} be the value of the optimal solution
    for all k **do**
      Solve pricing problem for commodity k to get path p_k with reduced cost $\bar{c}_{p_k}$
      if $\bar{c}_{p_k} < 0$ **then**
        Add p_k to $PATHS$
        Set $z^*_{LP} = z^*_{LP} + c_{p_k}$
      end if
    end for
    Select solver S from $SOLVERS$
    if solution found by S better than best_solution **then**
      Set best_solution = solution found by S
    end if
    Add new paths p from S to $PATHS$
    Take new cuts (π, π_o) from S, lift and add to $CUTS$
  end while

multiple solvers to choose from, we need a scheme for choosing a solver at each iteration. We use a scheme similar to the *roulette wheel* approach of [34] (see also [27]), i.e., we randomly pick solvers with the probability of choosing a solver favoring ones that have recently given improvement. Specifically, if C_t is the cost of the feasible solution F_t provided to solver S at iteration t and $C^{'}_t$ is the cost of the feasible solution $F^{'}_t$ found by S, then we calculate the improvement for solver S in that iteration as $D^S_t = C_t - C^{'}_t$ and in iteration T, we give

S the score

$$E_S = \sum_{t=1}^{T-1} \frac{1}{2^{T-t}} D_t^S.$$

The larger the value E_S is, the more effective solver S has been in improving solutions in recent iterations. Assuming m solvers, ordered such that $E_{S_i} \geq E_{S_{i+1}}$ for $i = 1, ..., m-1$, we assign selection probabilities

$$\pi_i = \frac{m - (i-1)}{\sum_{j=1}^{m} m - (j-1)}, \; i = 1, ..., m.$$

We do not use a simple proportional scheme so that solver S_i can be chosen even when $E_{S_i} = 0$, and to ensure that the selection is not biased too greatly in favor of solvers that have been effective in the last iterations.

2.3.1.1 Arc Subset Solves

Given a subset of arcs $A' \subseteq A$, the IP defined by A' is Arc-MCFCNF with A' replacing A. Our schemes for choosing the set A' are motivated by the simple idea that if we knew a set of arcs, $A^* = \{(i,j) \in A \mid y_{ij}^* = 1 \text{ for some optimal solution } (x^*, y^*)\}$, we could find the optimal solution to the complete instance by solving the arc subset IP for A^*. Hence our schemes use various measures to try and "guess" which arcs exist in an optimal solution.

Observe that if A' contains the arcs associated with the current best solution, then we can seed the arc subset IP with that solution and ensure that we only find solutions at least as good as the current best. Therefore, our schemes for choosing a subset A' start from the subset of arcs associated with the current best solution. Also, our schemes do not choose arcs directly but choose paths, thus ensuring that every arc chosen exists in some feasible solution. In fact, the schemes choose paths from among the paths that are part of the path-based formulation that is used to compute a dual bound. The schemes we next describe are used to generate a score for each path. We then randomly choose paths until $|A'| = N_A$, a preset threshold on $|A'|$, with the probability of choosing a path assigned so as to bias the selection towards paths with higher scores.

Scheme 1. This scheme is driven by the idea that paths that frequently appear in good solutions found during the course of the search may be in the optimal solution. Hence the

greater the number of times path p appears in an improving solution, the higher the score it is assigned.

Scheme 2. Recall that at each iteration we solve the LP relaxation of the path-based formulation with the set of paths $PATHS$. This scheme is motivated by the idea that the solution to the LP relaxation may guide us to paths that appear in the optimal IP solution. Let $x_p^{k,LP}$ represent the value of variable x_p^k in the most recent solution to the LP relaxation of our path-based formulation. The higher the value of $x_p^{k,LP}$, the higher the score given to path p.

Scheme 3. Recall that at each iteration we price new paths for each commodity for the LP-relaxation of the path-based formulation. This scheme is motivated by the idea that these paths are good candidates for the optimal IP solution. We also consider the reduced cost, $\bar{c}_p$, when assigning a score to a path, giving a higher score to a path that has lower reduced cost.

2.3.1.2 Commodity Subset Solves

Given the current solution $\bar{x}$ and a subset of commodities J, let $A' = \{(i,j) \in A \mid \bar{x}_{ij}^k = 0 \; \forall k \notin J\}$, i.e., the set of arcs that is *not* used by commodities *not* in J. Let

$$\tilde{f}_{ij} = \begin{cases} f_{ij} & (i,j) \in A' \\ 0 & (i,j) \notin A' \end{cases}$$

and $\tilde{u}_{ij} = u_{ij} - \sum_{k \notin J} d^k \bar{x}_{ij}^k$. Then we solve an Arc-MCFCNF only considering commodities in J and with $\tilde{f}_{ij}$ instead of f_{ij} and $\tilde{u}_{ij}$ instead of u_{ij}. That is, if there exists a commodity not in J that already uses an arc, we allow the commodities in J to use that arc for free (in terms of the associated fixed cost charge), but we make sure that the available capacity on the arc reflects the flow already on it.

Whereas we motivated our arc selection methods by the idea of guessing the optimal set of arcs, we motivate many of our commodity selection methods by the idea that when fixed charges (f_{ij}) are high, the optimal solution is likely to include as few arcs as possible. Hence we want to choose a subset of commodities J that are likely to share arcs.

Constructing a feasible solution to the full MCFCNF instance starting from the solution to the IP requires updating the paths used by the commodities in J and the appropriate y_{ij} variables for $(i,j) \in A'$. Note that since we do not have variables y_{ij} for $(i,j) \in A \setminus A'$ our subproblem cannot "turn off" an arc used by a commodity not in J. We can again seed the IP with the paths used by the commodities in J in the current best solution. The first three schemes we describe below are used to select a subset of commodities, J. The last three schemes are similar to the path schemes described above in that they assign a score to each commodity and then randomly choose commodities with a bias towards those with better scores. Unless otherwise noted, we always select a subset of commodities, J of fixed size, N_K.

Scheme 1. Again, recall that at each iteration we solve the LP relaxation of the path-based formulation with the set of paths $PATHS$. This scheme selects commodities whose paths in the current best solution contain arcs (i,j) for which the reduced cost of variable y_{ij}, denoted by $\bar{f}_{ij}$, are far from 0. In an LP solution complementary slackness implies $y_{ij} = 1$ only if $\bar{f}_{ij} = 0$. Thus the motivation for this scheme is to re-route commodities away from arcs that are far away from the complementary slackness condition.

Scheme 2. Suppose in the current best solution $(\bar{x}, \bar{y})$, there is a node v with three commodities $C1, C2, C3$ entering on a single arc, but leaving on three different arcs The motivation for this scheme is that when the arcs leaving node v are not fully used, these three commodities are good candidates for re-optimization. (Of course, a similar situation occurs when we have multiple commodities entering on different incoming arcs, but leaving on a single outgoing arc.) Assume we have a pre-defined utilization threshold $UTIL\_THRESHOLD$, where we calculate the utilization of an arc for the current best solution $(\bar{x}, \bar{y})$ as $UTIL_{ij} = \sum_{k \in K} \bar{d}^k \bar{x}^k_{ij} / u_{ij}$. Then we can score each node based on the number of under-utilized inbound and outbound arcs in the current best solution. Given those scores, we generate probabilities for selecting a node v. After randomly selecting a node v, we choose the commodities k such that in solution $(\bar{x}, \bar{y}), \exists w$ such that $\bar{x}^k_{v,w} > 0$ or $\bar{x}^k_{w,v} > 0$. Note this scheme may actually choose fewer (or more) than N_K.

Scheme 3. Since we want to find sets of commodities that are likely to share arcs, we search for commodities whose paths in the current best solution are close together. Given a randomly chosen node v, we perform breadth-first search with v as root. As we discover nodes visited by commodities k in our current best solution, we select the commodities. We continue the search until we have chosen N_K commodities.

Scheme 4. Suppose the network is very large and there are two commodities k and l that have only one feasible path from their source to their sink, say p_k and p_l. In that case, it is unlikely that these paths have a common arc. Hence re-optimizing commodities k and l together is not likely to lead to an improving solution. Conversely, if k and l have many paths then they are likely to share arcs and thus are good candidates to optimize together. While we do not know the set of all feasible paths for each commodity, during the course of the algorithm we do generate a set of feasible paths $\bar{P}(k)$ for each commodity k. Hence, the higher the number $|\bar{P}(k)|$, the higher the score assigned to commodity k.

Scheme 5. So far the selection schemes have been geared towards improving the current best solution. The purpose of this scheme is to provide diversification for our arc subset solvers. If there is a commodity k for which we have only generated a single path so far, then an arc subset solver will have little flexibility for re-routing it. Therefore, the lower the number $|\bar{P}(k)|$, the higher the score assigned to commodity k.

Scheme 6. The final scheme is specifically designed for instances of MCFCNF in which a commodity must be routed along a single path. The scheme selects commodities whose demand is split across many paths in the most recent solution to the LP relaxation of the path-based formulation. Again, define $x_p^{k,LP}$ to be the value of variable x_p^k in the solution of the LP relaxation to the path-based formulation and let $P'(k) = \{p \in \bar{P}(k) : 0 < x_p^{k,LP} < 1\}$. Then the higher the number $|P'(k)|$, the higher the score assigned to commodity k. In a sense, we are "repairing" commodities that violate the single path constraint in the solution to the LP relaxation of the path-based formulation.

2.3.2 Lower Bounds

At each iteration our solution approach produces both an upper and a lower bound. The lower bound is provided by the value of the LP relaxation of the path-based formulation. However, since the pricing problem is a shortest path problem, the bound is no better than the LP relaxation of the arc-based formulation, which is known to be weak. We attempt to strengthen the bound by adding valid inequalities before solving the path-based formulation. We first note that we can translate inequalities between the arc and path formulations using the transformation $x_{ij}^k = \sum_{p \in \bar{P}(k):(i,j) \in p} x_p^k$, where $\bar{P}(k)$ is the set of paths generated for commodity k thus far. Since we want valid inequalities that are violated by the optimal solution to the LP relaxation of the path-based formulation given the sets $\bar{P}(k)$, a desirable property of a valid inequality (π, π_0) is to have $\pi_{ij}^k > 0$ when $x_{ij}^k > 0$ in the optimal LP solution. We estimate the likelihood of x_{ij}^k being greater than 0 in the optimal LP solution by considering the ratio

$$r_{ij}^k = \frac{|\{p \in \bar{P}(k) : (i,j) \in p\}|}{|\bar{P}(k)|}$$

which calculates the percentage of paths we have generated (and hence will consider in the LP) for commodity k that use arc (i, j).

We have focused on two techniques for strengthening the path-formulation LP

- Judiciously adding disaggregate coupling inequalities, and
- Lifting cuts found while solving small IPs.

We could further strengthen the path-formulation bound by generating valid inequalities after solving the LP relaxation which the current fractional solution does not satisfy. However, identifying the separating inequality and re-solving the LP relaxation would would take time away from the search for primal solutions. This trade-off could be avoided in a parallel implementation by having separate processes work on the lower and upper bounds.

2.3.2.1 Disaggregate Inequalities

It is well-known that the disaggregate coupling inequalities (10) strengthen the LP bound, yet due to the great number of them it is typically not possible to add them all. In addition,

while adding the disaggregate coupling inequalities strengthens the LP bound they may also increase the solve time for the LP relaxation. Of course adding the inequality (10) for commodity k and arc (i,j) only strengthens the LP bound if in fact

$$x_{ij}^k = \sum_{p \in \bar{P}(k):(i,j) \in p} x_p^k > 0$$

in the optimal LP solution. Hence we determine the arcs (i,j) and commodities k for which the disaggregated coupling constraints will be added using the ratio r_{ij}^k. A threshold value $T \in [0,1]$ is specified and the inequality is added for commodity k and arc (i,j) only when $r_{ij}^k \geq T$.

2.3.2.2 Lifted Cover Inequalities

We have investigated re-using cuts (π, π_0) found while solving small IPs during the course of the algorithm. These cuts are lifted to make them globally valid and to strengthen them. As the inequalities are found while solving small IPs during the course of the algorithm, the computational effort is only in the lifting process. After lifting, these inequalities are added to the path formulation to strengthen the bounds its LP relaxation provides.

We have focused our efforts on the single-path variant of MCFCNF and investigated lifting cover inequalities based on arc capacities. A *cover* with respect to an arc (i,j) is a set $C \subseteq K$ such that $\sum_{k \in C} d^k > u_{ij}$. Since the variables x_{ij}^k are binary for the single-path variant we have the *cover inequality* $\sum_{k \in C} x_{i,j}^k \leq |C| - 1$.

We use the procedure suggested by [18] for separating cover inequalities. Note that since the cover inequality is generated during the solution of a commodity subset IP some variables are fixed to either 0 or 1. As suggested by [18], we first uplift variables fixed to 0 and then down-lift those fixed to 1.

It is well known that the strength of the lifted cover inequality depends on the order in which the variables fixed to 0 (or 1) are lifted. Therefore we first lift variables x_{ij}^k whose value is likely to be positive in the optimal LP solution. Hence, the lifting order is quite simple: lift in decreasing order of r_{ij}^k. The complete procedure is outlined in Algorithm 4.

Currently, we limit ourselves to cover inequalities that are violated in the small IP.

Algorithm 4 Cover Inequality Lifting

Require: Cover Inequality (π, π_0) for arc (i, j) from commodity subset IP
Require: Set of commodities K^0 such that when generating (π, π_0), x_{ij}^k fixed to 0
Require: Set of commodities K^1 such that when generating (π, π_0), x_{ij}^k fixed to 1
Calculate $r_{ij}^k = \frac{|\{p \in \bar{P}(k):(i,j) \subseteq p\}|}{|\bar{P}(k)|}, \ \ \forall k \in K^0 \cup K^1$
Uplift variables $x_{ij}^k, k \in K^0$ in descending order of r_{ij}^k
Downlift variables $x_{ij}^k, k \in K^1$ in descending order of r_{ij}^k
return new cut $(\pi^{'}, \pi_0^{'})$

Given that we are especially interested in strengthening the LP relaxation of the path-based formulation, this methodology can be extended to search for cover inequalities that are *not* violated in the small IP, but are likely to be violated when all commodities are considered.

Since our lifting order depends on the sets $\bar{P}(k)$ which may change at each iteration, we could re-lift the cover inequalities as the path sets change and get different valid inequalities. However, we do not re-lift because of our time-constrained setting and the computational effort lifting requires.

2.3.3 Initial Feasible Solution

Our scheme for generating an initial feasible solution has two phases. In Phase I we find a path for each commodity by solving a shortest path problem where arc costs reflect a linearized fixed charge and are updated using a slope-scaling approach. Heuristics based on updating the linearization of the fixed charge are given in [4], [11], and [22]. The solution to the LP relaxation of Arc-MCFCNF has $y_{ij} = \frac{1}{u_{ij}} \sum_k d^k x_{ij}^k$, hence we have $f_{ij} y_{ij} = \frac{f_{ij}}{u_{ij}} \sum_k d^k x_{ij}^k = \sum_k f_{ij} \frac{d^k}{u_{ij}} x_{ij}^k$. This suggests that we can use a variable cost $\tilde{c}_{ij} = c_{ij} d^k + f_{ij} \frac{d^k}{u_{ij}}$ for each arc (i, j) and then solve a shortest path problem using these arc costs for each commodity k. Instead, we recognize that by assigning paths to commodities sequentially we remove the capacity of some arcs. Hence we update the variable cost on an arc (i, j) to reflect how much of its capacity remains. The details are given in Algorithm 5.

If we have a feasible solution after Phase I, we stop. Since we do not explicitly enforce the capacity constraints on arcs during Phase I, we may not end with a feasible solution. In this case, we go on to Phase II, where for each arc whose capacity is exceeded, we find a set

of commodities to re-route away from the arc in an attempt to "fix the infeasibility." The details are given in Algorithm 6. Note that this method cannot guarantee that a feasible solution is found when one exists, nor can it detect whether an instance is infeasible.

Algorithm 5 Phase I algorithm

Sort commodities in descending order of demand d^k
Set $\tilde{u}_{ij} = u_{ij}$
for $k = 1$ to K **do**
  Set $\tilde{c}_{ij} = c_{ij} + f_{ij} * \frac{d^k}{\tilde{u}_{ij}}$
  Solve a shortest path problem for commodity k w.r.t. arc costs $\tilde{c}_{ij}$; let x_{ij}^k indicate when arc (i, j) used by k
  Set $\tilde{u}_{ij} = max(1, \tilde{u}_{ij} - x_{ij}^k * d^k)$
end for

Algorithm 6 Phase II algorithm

while $\exists (i, j) : \sum_k x_{ij}^k * d^k > u_{ij}$ **do**
  Randomly choose such an (i, j)
  Sort commodities using the arc $(x_{ij}^k = 1)$ in ascending order of demand d^k
  Find the smallest l such that $\sum_{k=l}^{K} x_{ij}^k * d^k \leq u_{ij}$
  Solve our commodity subset IP for those first $l - 1$ commodities
end while

2.4 Computational Results

There are many questions to answer regarding the primal and dual-side performance of our approach, which we refer to as IP Search. Specifically:

- Primal-Side
  - How competitive is IP Search with existing metaheuristics and commercial IP solvers?
  - Is there value in using both commodity subset and arc subset IPs?
  - Do the solvers contribute equally to finding good solutions?
- Dual-Side
  - Does IP Search produce a stronger dual bound than the optimal solution to the LP relaxation of the arc-based formulation?

- Is the dual bound produced by IP Search competitive with the root node bound produced by a commercial IP solver using an arc-based formulation?
- Do the disaggregate inequalities strengthen the dual bound?
- Do the lifted cover inequalities strengthen the dual bound?

IP Search is implemented in C++ and uses CPLEX 9.1 to solve the subset IPs. Other than those taken from the literature, all the results are obtained from experiments performed on a machine with 8 Intel Xeon CPUs running at 2.66 GHz and 8 GB RAM. Whenever CPLEX 9.1 is used as a benchmark, it was run with an emphasis on integer feasibility, with the disaggregate constraints $x_{ij}^k \leq y_{ij}$ added as user cuts, and with an optimality tolerance set to 5%. All computation times reported are in seconds.

2.4.1 Instance Generation

A network generator was used to create instances for experimentation. The generator creates instances without parallel arcs. The inputs to the generator are:

- The number of nodes, arcs, and commodities in the network
- A range $[c_l, c_u]$ for arc variable costs
- A range $[b_l, b_u]$ for commodity quantities
- A range $[1, k_u]$ of how many commodities can fit on an arc
- A ratio R of arc fixed cost to variable cost

The generator creates an arc (i, j) by randomly picking nodes i and j. The variable cost c_{ij} for arc (i, j) is uniformly drawn from $[c_l, c_u]$. The fixed cost f_{ij} for arc (i, j) is uniformly drawn from $[Rc_l, Rc_u]$. The capacity u_{ij} for an arc is set to kb, where k is uniformly drawn from $[1, k_u]$ and b is uniformly drawn from $[b_l, b_u]$. Commodities k are created by randomly picking pairs of nodes $(o(k), d(k))$ such that a path exists between the two nodes. The quantity b_k for commodity k is uniformly drawn from $[b_l, b_u]$. Finally all commodity and arc data is rounded down to integer values.

We use the same instance classifications as [15]. An instance is classified as F if the ratio of fixed to variable cost for an arc is high and V otherwise. An instance is classified as T if the approximate number of commodities that can be accommodated on an arc is low and L otherwise. Hence we have 4 classes of instances $(F,T),(F,L),(V,T),(V,L)$.

2.4.2 Calibration

Since we solve IPs as subroutines in IP Search, their size is important. If the IP is too large we may not find a good improving solution in the time alloted to the IP solver. On the other hand, if the IP is too small it simply may not yield a large enough neighborhood to contain a good improving solution. We determine the arc and commodity subset sizes for a specific instance size by measuring the performance of the heuristic as the subset size grows. For example, to determine the number of commodities in our subset when running IP Search on instances with 500 nodes, 2500 arcs and 150 commodities we randomly generate 5 instances of that size. We then run the heuristic on each of the 5 instances multiple times, varying the number of commodities selected.

In the course of our calibration experiments we observed that choosing 20% of the total number of commodities in the instance is a reasonable rule-of-thumb. It is not surprising that the size of the commodity subset grows with the number of commodities in the instance. The more commodities outside of the subset, the more likely an arc (i,j) is used by a commodity outside of the subset, which implies there will be no fixed charge for arc (i,j). Hence the larger the number of commodities in the instance the more the commodity subset IPs become multi-commodity flow problems without fixed charges.

2.4.3 Upper Bound

2.4.3.1 Comparison to Tabu Cycle and Path Relink

To examine how IP Search compares with existing metaheuristics, we compared it to a cycle-based tabu-search algorithm [15] and a cycle-based path-relinking approach [16] on the instances used in [16]; except for the 6 instances identified as "easy" (solved in a few seconds by a commercial IP solver). Note that the comparison is only for the variant in which commodities can be routed along multiple paths (the variant handled by the

metaheuristics). The results are shown in Table 1, where we report the instance, the value of the solution found by CPLEX in 12 hours (CPLEX), the value of the solution obtained by the tabu search algorithm (Tabu Cycle), the value of the solution obtained by the path relinking algorithm (Path Relink), the value of the solution obtained by IP Search (IP Search), the percentage difference between the value of the solution found by IP Search and the value of the solution found by CPLEX in 12 hours (CPLEX gap), the percentage difference between the value of the solution found by IP Search and the best of the values of the solutions found by the tabu search and path relinking algorithms (Best gap), the optimality gap (Opt gap), and for each of the different algorithms the time to reach the best solution (TTB). When we report a percentage difference (or gap) between IP Search and "X", it is computed as $100\frac{\text{IP Search}-\text{X}}{\text{IP Search}}$. To compute the the optimality gap, we use the dual bound produced by CPLEX in 12 hours. Note that we imposed a time limit of 15 minutes (900 seconds) on IP Search.

We first observe that IP Search finds a better solution than the metaheuristics in all but 2 of the 37 instances. Since different hardware platforms were used to produce the metaheuristic and IP Search results a comparison of computing times cannot be exact. However, the time to the best solution for instance (30,700,400,F,L) indicates that the computing time is orders of magnitudes longer for the metaheuristics than for IP Search. For the instance (100,400,10,F,T) the percentage difference between the value of the best of the metaheuristics solutions and the value of the solution produced by IP Search is less than 1. Hence, we can comfortably conclude that IP Search is superior to these metaheuristics. We next observe that IP Search is producing solutions that are very competitive with those produced by CPLEX in 12 hours. Even though the instances are relatively small, we see that CPLEX often needs more than 5 hours to find its best solution. Finally, we observe that the optimality gap for the solution produced by IP Search is small, within 5% for 24 of the 37 instances and only 3.96% on average.

Table 1: Comparison with Tabu Search and Path Relinking

	Solution Value							TTB			
Problem	CPLEX	Tabu Cycle	Path Relink	IP Search	CPLEX Gap	Best Gap	Opt Gap	CPLEX	Tabu Cycle	Path Relink	IP Search
100,400,10,V,L	28,423	28,786	28,485	28,423	0.00	-0.22	0.00	3	252	89	35
100,400,10,F,L	23,949	24,022	24,022	23,949	0.00	-0.30	0.00	3,354	196	82	9
100,400,10,F,T	**63,764**	**67,184**	**65,278**	**65,885**	3.22	**0.92**	**13.97**	**34,301**	**451**	**209**	**813**
100,400,30,V,T	384,802	385,508	384,926	384,836	0.01	-0.02	0.60	1,290	1,199	492	330
100,400,30,F,L	49,018	51,831	51,325	49,694	1.36	-3.28	3.44	22,457	717	314	886
100,400,30,F,T	138,948	147,193	141,359	141,365	1.71	0.00	11.65	36,001	1,300	480	888
20,230,40,V,L	423,848	430,628	424,385	424,385	0.13	0	0.13	1	214	148	4
20,230,40,V,T	371,475	372,522	371,811	371,779	0.08	-0.01	0.08	1	241	156	41
20,230,40,F,T	643,036	652,775	645,548	643,187	0.02	-0.37	0.02	13	259	172	45
20,230,200,V,L	94,213	100,001	100,404	95,097	0.93	-5.16	6.21	6,118	2,585	2,494	822
20,230,200,F,L	137,642	148,066	147,988	141,253	2.56	-4.77	8.72	8,079	3,142	2,878	691
20,230,200,V,T	97,914	106,868	104,689	99,410	1.50	-5.31	4.9	7,097	2,730	2,210	821
20,230,200,F,T	136,817	147,212	147,554	140,273	2.46	-4.95	8.06	17,397	3,634	3,386	156
20,300,40,V,L	429,398	432,007	429,398	429,398	0.00	0	0	1	305	224	19
20,300,40,F,L	586,077	602,180	590,427	586,077	0.00	-0.74	0	5	336	228	29
20,300,40,V,T	464,509	466,115	464,509	464,509	0.00	0	0	2	379	247	24
20,300,40,F,T	604,198	615,426	609,990	604,198	0.00	-0.96	0	1	350	214	68
20,300,200,V,L	74,929	81,367	78,184	75,319	0.52	-3.8	3.75	36,364	4,086	3,566	802
20,300,200,F,L	117,180	122,262	123,484	117,543	0.31	-4.01	7.18	42,220	4,210	4,012	686
20,300,200,V,T	74,991	80,344	78,866	76,198	1.58	-3.5	3.44	6,490	4,204	3,924	388
20,300,200,F,T	108,850	113,947	113,584	110,344	1.35	-2.94	7.08	42,585	4,855	3,857	396
30,520,100,V,L	53,958	56,603	54,904	54,113	0.29	-1.46	0.82	436	2,261	1,194	218
30,520,100,F,L	94,264	103,657	102,054	94,388	0.13	-8.12	7.78	39,409	2,684	1,459	226
30,520,100,V,T	52,048	54,454	53,017	52,174	0.24	-1.62	1.34	38,260	2,716	1,513	455
30,520,100,F,T	98,358	105,130	106,130	98,883	0.53	-6.32	5.97	39,302	2,892	1,522	815
30,520,400,V,L	113,650	122,673	119,416	114,042	0.34	-4.71	2.85	36,436	55,771	27,477	394
30,520,400,F,L	152,418	164,140	163,112	154,218	1.17	-5.77	5.79	30,624	40,070	36,669	750
30,520,400,V,T	114,799	122,655	120,170	114,922	0.11	-4.57	0.97	31,588	4,678	23,089	621
30,520,400,F,T	154,831	169,508	163,675	154,606	-0.15	-5.87	3.72	31,127	49,886	52,173	466
30,700,100,V,L	47,603	50,041	48,723	47,612	0.02	-2.33	0.02	205	2,959	1,860	32
30,700,100,F,L	59,995	64,581	63,091	60,700	1.16	-3.94	6.22	18,900	3,182	1,837	741
30,700,100,V,T	45,879	48,176	47,209	46,046	0.36	-2.53	2.21	34,111	3,746	1,894	371
30,700,100,F,T	54,904	57,628	56,575	55,609	1.27	-1.74	4.78	39,396	3,547	1,706	387
30,700,400,V,L	98,992	107,727	105,116	98,718	-0.28	-6.48	2.93	40,901	38,857	22,314	222
30,700,400,F,L	**140,618**	**150,256**	**145,026**	**152,576**	7.84	**4.95**	**15.42**	**10,231**	**68,214**	**75,664**	**860**
30,700,400,V,T	96,139	101,749	101,212	96,168	0.03	-5.24	2.81	37,661	51,764	24,288	365
30,700,400,F,T	131,946	144,852	141,013	131,629	-0.24	-7.13	3.79	5,464	79,053	44,936	225
Average					.87	-2.76	3.96	18860.30	12106.08	9431.81	408.14

2.4.3.2 Comparison to Commercial IP Solver

Next we focus on the performance of IP Search on large instances for the variant in which each commodity has to be routed along a single path by comparing the value of the solution produced by IP Search to the value of the solution produced by CPLEX in 15 minutes (CPLEX-15M) and in 12 hours (CPLEX-12H). We generated 24 instances of the classes (F,T) and (F,L), which we refer to as the GT instances. The results are given in Table 2. An "X" in a column indicates that no feasible solution was found. As before, the value reported for IP Search is the value of the solution produced within 15 minutes.

We observe that in *every* instance IP Search finds a better solution in 15 minutes than CPLEX does in 12 hours. We also see that the improvement over the solution found by CPLEX in 15 minutes is significant, often greater than 30%. Even the improvement over the solution found by CPLEX in 12 hours is impressive, often greater than 20%. Unfortunately, little can be said with confidence regarding the true optimality gap of the solutions produced by IP Search since the dual bounds produced by CPLEX change very little over the course of the execution and are likely to be weak. In fact, for many of the loosely capacitated instances, CPLEX did not find a significantly better primal solution in 12 hours than it did in 15 minutes. This highlights the difficulty that an LP-based branch-and-bound algorithm can have in finding good primal solutions when the dual bounds are weak.

We have also observed that IP search outperforms CPLEX in very little time. In fact, for the GT instances, IP search needed on average less than 1 minute to produce a solution that is better than the best solution produced by CPLEX in 15 minutes and less than 3 minutes to produce a solution that is better than the best solution produced by CPLEX in 12 hours.

The time limit of 15 minutes on IP Search is, of course, self-imposed. Table 3 reports the results for IP Search after 15, 30, and 60 minutes as well as the percentage improvement over the initial solution found. We see that the greatest percentage improvement occurs in the first 15 minutes, but that IP Search continues to find improving solutions when given more time. The column labeled "Times Phase II" gives the number of times Phase II of the scheme for finding an initial feasible solution is executed. Not surprisingly, Phase II is only

Table 2: Primal-side Comparison with CPLEX

Problem	CPLEX-15M	CPLEX-12H	IP Search	CPLEX-15M Gap	CPLEX-12H Gap	Opt Gap
500,2000,50,F,L	5,301,081	5,301,081	3,910,120	-35.57	-35.57	55.94
500,2000,50,F,T	X	7,927,065	5,249,040	N/A	-51.02	31.14
500,2000,100,F,L	8,944,724	8,299,799	6,764,310	-32.23	-22.70	63.52
500,2000,100,F,T	10,199,000	8,306,181	7,718,750	-32.13	-7.61	31.84
500,2000,150,F,L	10,996,000	10,080,000	8,618,060	-27.59	-16.96	63.78
500,2000,150,F,T	12,115,000	10,770,000	9,448,890	-28.22	-13.98	65.32
500,2000,200,F,L	13,808,000	12,824,000	10,333,200	-33.63	-24.10	64.37
500,2000,200,F,T	X	X	12,425,600	N/A	N/A	49.63
500,2500,50,F,L	4,611,275	4,611,275	3,841,350	-20.04	-20.04	70.88
500,2500,50,F,T	5,779,926	5,084,529	4,666,740	-23.85	-8.95	32.56
500,2500,100,F,L	9,351,042	9,251,042	6,875,420	-36.01	-34.55	71.97
500,2500,100,F,T	9,724,997	7,995,284	7,235,520	-34.41	-10.50	46.31
500,2500,150,F,L	13,660,000	12,497,000	9,730,100	-40.39	-28.44	77.45
500,2500,150,F,T	11,385,000	10,683,000	7,934,360	-43.49	-34.64	72.22
500,2500,200,F,L	15,539,000	13,468,000	11,261,300	-37.99	-19.60	75.02
500,2500,200,F,T	18,906,000	14,948,000	12,825,300	-47.41	-16.55	58.30
500,3000,50,F,L	5,098,318	5,098,318	3,596,980	-41.74	-41.74	76.27
500,3000,50,F,T	5,615,096	4,866,768	4,504,260	-24.66	-8.05	41.85
500,3000,100,F,L	8,721,798	8,721,798	6,577,980	-32.59	-32.59	75.06
500,3000,100,F,T	10,119,000	8,330,109	7,517,970	-34.60	-10.80	49.67
500,3000,150,F,L	12,628,000	12,623,000	9,214,960	-37.04	-36.98	80.31
500,3000,150,F,T	12,615,000	10,147,000	9,186,840	-37.32	-10.45	55.71
500,3000,200,F,L	15,039,000	13,441,000	10,853,400	-38.56	-23.84	80.03
500,3000,200,F,T	17,883,000	13,674,000	11,578,000	-54.46	-18.10	58.35
Average				-35.18	-22.95	60.31

executed for tightly-capacitated instances. In fact, for the majority of instances, Phase II is not needed at all.

Table 3: IP Search Given More Time

Problem	Times Phase II	Init. Feas. Soln	15 Minute Soln	30 Minute Soln	60 Minute Soln	Imp. After 15 M	Imp. After 30 M	Imp. After 60M
500,2000,50,F,L	0	6,175,840	3,910,120	3,907,440	3,823,610	-36.69	-36.73	-38.09
500,2000,50,F,T	5	6,314,980	5,249,040	5,112,490	4,949,780	-16.88	-19.04	-21.62
500,2000,100,F,L	0	10,244,000	6,764,310	6,655,370	6,453,880	-33.97	-35.03	-37.00
500,2000,100,F,T	4	10,125,700	7,718,750	7,632,220	7,619,670	-23.77	-24.63	-24.75
500,2000,150,F,L	0	13,317,800	8,618,060	8,279,270	8,081,600	-35.29	-37.83	-39.32
500,2000,150,F,T	0	14,557,800	9,448,890	9,250,760	8,807,650	-35.09	-36.45	-39.50
500,2000,200,F,L	0	17,411,800	10,333,200	10,138,900	9,828,350	-40.65	-41.77	-43.55
500,2000,200,F,T	0	17,543,000	12,425,600	12,117,800	11,893,100	-29.17	-30.93	-32.21
500,2500,50,F,L	0	5,712,380	3,841,350	3,780,160	3,612,030	-32.75	-33.83	-36.77
500,2500,50,F,T	1	5,898,160	4,666,740	4,661,160	4,600,200	-20.88	-20.97	-22.01
500,2500,100,F,L	0	10,640,700	6,875,420	6,621,140	6,400,140	-35.39	-37.78	-39.85
500,2500,100,F,T	1	10,077,400	7,235,520	7,065,020	6,953,660	-28.20	-29.89	-31.00
500,2500,150,F,L	0	14,834,200	9,730,100	9,475,760	9,089,920	-34.41	-36.12	-38.72
500,2500,150,F,T	0	13,365,000	7,934,360	7,673,410	7,571,640	-40.63	-42.59	-43.35
500,2500,200,F,L	0	17,446,900	11,261,300	10,623,700	10,099,200	-35.45	-39.11	-42.11
500,2500,200,F,T	1	18,628,100	12,825,300	11,843,200	11,452,900	-31.15	-36.42	-38.52
500,3000,50,F,L	0	5,687,990	3,596,980	3,519,400	3,457,280	-36.76	-38.13	-39.22
500,3000,50,F,T	0	5,553,000	4,504,260	4,405,120	4,262,350	-18.89	-20.67	-23.24
500,3000,100,F,L	0	10,383,600	6,577,980	6,202,380	6,015,950	-36.65	-40.27	-42.06
500,3000,100,F,T	2	10,350,000	7,517,970	7,444,950	7,186,810	-27.36	-28.07	-30.56
500,3000,150,F,L	0	13,707,300	9,214,960	9,126,970	8,919,720	-32.77	-33.42	-34.93
500,3000,150,F,T	0	12,958,300	9,186,840	8,898,880	8,709,390	-29.10	-31.33	-32.79
500,3000,200,F,L	0	16,869,900	10,853,400	10,419,400	10,040,000	-35.66	-38.24	-40.49
500,3000,200,F,T	1	17,071,600	11,578,000	10,750,200	10,390,700	-32.18	-37.03	-39.13
Average						-31.66	-33.59	-35.45

We observed in Table 2 that IP Search significantly outperforms CPLEX on large instances when each commodity has to be routed along a single path. However the dual bounds produced by CPLEX were too weak to get a sense of the real quality of the solutions produced by IP Search. Next we investigate what happens when we have smaller instances and require a commodity to be routed along a single path. The results are given in Table 4. As before, IP Search is limited to 15 minutes. When CPLEX obtains a better solution than IP Search, we report how long it took CPLEX to find the first solution that is better than the solution found by IP Search (CPLEX Time Beat). We observe that in only 2 of the 37 instances does CPLEX find a better solution in 15 minutes than IP Search. In those 2 instances the difference in solution value between IP Search and CPLEX is less than .15%. On the other hand, IP Search often finds solutions that are 5% better than what CPLEX finds in 15 minutes. The optimality gap information reveals that the solutions produced by IP Search are of high quality, on average within 4% of optimality. When CPLEX does find a better solution, it often requires more than 3 hours to do so. We also see that IP Search produces solutions within 15 minutes that are competitive with what CPLEX produces in 12 hours.

Table 4: Primal Side Comparison with CPLEX - Metaheuristic Instances - Single Path

Problem	CPLEX-15M	CPLEX-12H	IP Search	CPLEX-15M Gap	CPLEX-12H Gap	Opt Gap	CPLEX Time Beat
100,400,10,V,L	31,785	31,785	31,785	0	0	0	-
100,400,10,F,L	24,104	24,104	24,104	0	0	0	-
100,400,10,F,T	328,177	328,177	328,177	0	0	0	-
100,400,30,V,T	750,420	750,420	750,420	0	0	0	-
100,400,30,F,L	55,665	50,391	52,779	-5.47	4.52	6.90	10,998
100,400,30,F,T	188,113	188,113	188,113	0	0	0	-
20,230,40,V,L	423,933	423,933	424,671	0.17	0.17	0.17	1
20,230,40,V,T	398,870	398,870	398,870	0	0	0	-
20,230,40,F,T	668,699	668,699	668,699	0	0	0	-
20,230,200,V,L	102,118	95,126	96,456	-5.87	1.38	5.18	2,595
20,230,200,F,L	160,535	141,197	141,700	-13.29	0.35	6.37	-
20,230,200,V,T	107,145	100910	100,884	-6.21	-0.03	6.4	24,469
20,230,200,F,T	157,874	143781	141,734	-11.39	-1.44	8.98	-
20,300,40,V,L	430,253	430,253	430,253	0	0	0	-
20,300,40,F,L	597,059	597,059	597,059	0	0	0	-
20,300,40,V,T	501,766	501,766	501,766	0	0	0	-
20,300,40,F,T	643,395	643,395	643,395	0	0	0	-
20,300,200,V,L	81,101	78262	76,946	-5.4	-1.71	5.7	-
20,300,200,F,L	128,871	120301	119,590	-7.76	-0.59	8.14	-
20,300,200,V,T	78,900	77303	77,858	-1.34	0.71	4.99	17,403
20,300,200,F,T	120,997	110897	110,609	-9.39	-0.26	6.1	20,911
30,520,100,V,L	54,394	54,387	54,454	0.11	0.12	0.12	308
30,520,100,F,L	103,771	98797	97,199	-6.76	-1.64	6.34	-
30,520,100,V,T	53,812	53,812	53,812	0	0	0	-
30,520,100,F,T	102,186	99204	100,871	-1.3	1.65	4.35	8,677
30,520,400,V,L	132,799	120781	117,078	-13.43	-3.16	5.25	-
30,520,400,F,L	164,469	159146	157,682	-4.3	-0.93	7.92	-
30,520,400,V,T	132,935	120326	118,214	-12.45	-1.79	3.63	-
30,520,400,F,T	166,561	158396	161,577	-3.08	1.97	7.87	22,897
30,700,100,V,L	47,883	47,883	47,883	0	0	0	-
30,700,100,F,L	63,275	60856	61,254	-3.3	0.65	3.43	12,109
30,700,100,V,T	47,864	47,670	47,736	-0.27	0.14	0.14	1,236
30,700,100,F,T	57,218	56686	57,125	-0.16	0.77	0.77	2,019
30,700,400,V,L	103,839	100332	102,216	-1.59	1.84	6.28	12,599
30,700,400,F,L	677,227	151043	144,077	-370.05	-4.83	10.48	-
30,700,400,V,T	100,937	99972	99,118	-1.84	-0.86	5.52	-
30,700,400,F,T	142,604	137149	138,787	-2.75	1.18	8.68	11,605
Average				-13.17	-0.05	3.51	10,559.07

2.4.3.3 Value of Subset Solvers

Given the success of IP search it is natural to ask which of its components are responsible. Since IP search utilizes two different neighborhoods, one defined by a subset of arcs and the other by a subset of commodities, the first question is what if we only used one of them? To answer this question we ran IP search on the GT instances two more times; the first time only using arc subset IPs and the second time only using commodity subset IPs. The percentage difference, or gap, information for only using subset type X is calculated as $100\frac{\text{X–Arc \& Commodity}}{\text{Arc \& Commodity}}$. We observed that on average, only using arc subset IPs lead to a gap of 6.67% whereas only using commodity subset IPs lead to a gap of 2.73%. Hence, while using both types of neighborhoods typically yields the best solution, each neighborhood is effective by itself.

To gain a greater understanding of the efficacy of each selection scheme, we examine in detail the contribution of each on finding improving solutions. Table 5 gives the breakdown by scheme, averaged over the GT instances, after running IP Search for 60 minutes. We report the percentage of iterations a scheme was executed (% Called) and the percentage improvement that can be attributed to a scheme (% Improvement). The schemes are denoted by A-x for arc subset selection scheme x and C-x for commodity subset selection scheme x. We observe that the vast majority of solution improvement is due to the commodity

Table 5: Selection Method Contribution

Scheme	% Called	% Total Improvement
A-1	6	3
A-2	4	2
A-3	9	8
C-1	14	13
C-2	10	4
C-3	13	9
C-4	14	16
C-5	14	28
C-6	16	17

subset IPs. This may simply be due to the biased random selection mechanism repeatedly selecting the commodity subset solvers because of their effectiveness. Commodity selection scheme 5 results in the largest percentage improvement. Recall that this is the scheme that selects commodities for which few paths have been discovered and was designed to aid in

diversification for the arc subset solvers. These results indicate the scheme is effective in finding improving solutions itself.

Recall we used a biased random selection method to determine which solver to execute at an iteration. To further evaluate the value of the biased random selection, we compare it to a simple round-robin approach for selecting solvers. Our simple round-robin scheme executes the solvers in the following order: A-1, C-1, C-2, A-2, C-3, C-4, A-3, C-5, and C-6. We executed IP search using the round-robin scheme on the GT instances. Using a gap value calculated as $100\frac{\text{Round Robin}-\text{Biased Random}}{\text{Round Robin}}$, we observed that the average gap is 2.46%, indicating that biased random selection does better than round-robin, but not significantly better.

2.4.4 Lower Bound

The next question we address is the strength of the lower bound produced by IP Search. Since we want to evaluate the value of the lifted cover inequalities, we only consider the single-path variant of MCFCNF. For the GT instances we compared the value of the optimal solution to the LP-relaxation (LPR), the value of the root node bound (Root Node) produced by CPLEX,the lower bound IP Search produces and the percentage differences between the bound IP Search produces and the other two bounds (LP Gap and Root Gap). The percentage differences are calculated as $100\frac{\text{IP Search}-\text{LPR}}{\text{IP Search}}$ and $100\frac{\text{Root Node}-\text{IP Search}}{\text{Root Node}}$. Note that this means a positive percentage difference with the value of the LP Relaxation indicates that IP Search has given a stronger lower bound and a positive percentage difference with the value of the root node LP indicates that CPLEX has a stronger root node bound. Since the disaggregate coupling constraints are added as user cuts, they are not present in the LP relaxation, but are reflected in the root node bound in addition to any other cuts generated by CPLEX.

We observed that on average, LP Gap is 25% and hence IP Search produces lower bounds which are much stronger than the value of the LP Relaxation. Furthermore, we have observed that the difference is larger for loosely capacitated instances. On the other hand, we observed that Root Gap is 23%. Hence, we see that by adding cuts, CPLEX

produces a root node bound which is stronger than the bound produced by IP Search.

Since both disaggregate coupling constraints and lifted cover inequalities are used to strengthen the dual bound IP Search produces, we next analyzed their respective contributions. For the GT instances, we compare LPR, the bound produced by IP Search when only considering the disaggregated coupling constraints, Disaggr, and the bound produced by IP Search when considering the disaggregated coupling constraints and the lifted cover inequalities, Disaggr + Cover. We then calculated gaps similar to the LP Gap described above. We observed that the Disaggr Gap is 21%, indicating that by only adding the disaggregated coupling constraints we are able to derive a much stronger lower bound than LPR. Furthermore, we observed that the Disaggr +Cover Gap is 25% indicating that the lifting of inequalities found during the solution of subset IPs does contribute to strengthening the lower bound.

We also observed that with a threshold value $T = .25$ we typically only add 1% of the possible disaggregate coupling constraints. In addition, we observed that the lifted cover inequalities are most effective for tightly-capacitated instances.

While we are pleased by the success of our approach and in particular its ability to handle both large instances of MCFCNF and different problem variants, we also recognize that it is heavily dependent on the class of problem studied. Although the concept of using information from both the compact and extended formulation of a problem may be re-used in the design of heuristics for other problems, many of our schemes for choosing variable subsets are derived from our understanding of the structure of good solutions to instances of the MCFCNF. Thus we next try and understand to what extent we can develop schemes for choosing variable subsets that are not problem specific as well as whether we should limit ourselves to integer programs created by fixing variables.

CHAPTER III

MODELING IP-BASED SEARCH

When integer programming (IP) models are used in operational situations there is a need to consider the tradeoff between the conflicting goals of solution quality and solution time, since for many problems solving realistic-size instances to a tight tolerance is still beyond the capability of state-of-the-art solvers. However, by appropriately defining small instances, good primal solutions frequently can be found quickly.

This approach is taken, for example, within LP-based branch-and-bound algorithms using techniques such as local branching ([14]) and RINS ([12]). These techniques use information from the LP solution and incumbent solution to define a small IP, which is then optimized. These techniques can be applied to any integer program and are available in commercial solvers such as CPLEX.

Another approach to defining small instances is to use problem structure as in Chapter 2 where small IPs are chosen according to the attributes of previous solutions, such as the arcs used by various commodities. Combining exact and heuristic search techniques by solving small IPs has received quite a lot of attention recently, see, for example, [13], [36], [3], and [35].

Still another heuristic approach is to use structure to define neighborhoods that can be searched in polynomial time such as the very large scale neighborhood search approach of [2].

A key difference between the methods that are embedded in an LP-based tree search algorithm (local branching and RINS) and the others is that they are connected with a dual bounding procedure so that optimality or weaker tolerance gaps can be proved.

In this chapter we introduce a new approach to finding good solutions quickly that is capable of proving optimality as well. It is different from techniques such as local branching and RINS since it uses problem structure to define the small IPs to be solved. It is different

from other IP-based local search methods since the IPs to be solved are determined by a column generation scheme. The embedding of this column generation scheme into a branch-and-price algorithm gives the dual bounds that provide the capability of proving optimality.

Our extended formulation, which requires column generation, is very different from typical column generation formulations that employ structurally different objects from the compact formulation, for example, paths rather than arcs. Our extended formulation keeps the original variables from the compact formulation and augments them with an exponential number of additional variables that are used to define problem restrictions to obtain small IPs. By preserving the original compact formulation, we are able to enrich it by preprocessing, cutting planes or any other techniques normally used in a branch-and-cut framework.

The computational results demonstrate that the approach achieves its goals. For the instances used in the computational study, the approach often produces in 15 minutes a proven near-optimal primal solution. More specifically, the primal solution found in 15 minutes is often better than the one CPLEX produces in 6 hours, and the dual bound is usually close to the one CPLEX produces in 6 hours.

The remainder of this chapter is organized as follows. In Section 3.1 we present our extended formulation which models the search for optimal solutions to an IP. In Section 3.2 we present a parallelized branch-and-price approach for solving the model. In Section 3.3 we discuss how the approach is applied to the MCFCNF and in Section 3.4 we present the computational results of applying the approach to MCFCNF.

3.1 Modeling the Search for Optimal Solutions

For convenience, we present a model for binary mixed integer programs, but it can be used for general integer programs as well. Consider problem P given by

$$\begin{array}{lrcll} \max & cx & + & dy & \\ \text{s.t.} & Ax & + & By & = b \\ & & & x \text{ real}, & y \text{ binary}. \end{array} \tag{11}$$

Let V_P^* denote the optimal value of P, $S_P = \{(x, y) |\ Ax + By = b, x \text{ real}, y \text{ binary}\}$ be the set of feasible solutions to P, LP denote the linear relaxation of P and V_{LP}^* its optimal value.

For a given integer matrix N and a given integer vector q, both of appropriate dimension, we define *restriction* $P_N(q)$ of P as:

$$\begin{array}{lrcll} \max & cx & + & dy & \\ \text{s.t.} & Ax & + & By & = b \\ & & & Ny & \leq q \\ & & & x \text{ real}, & y \text{ binary} \end{array} \tag{12}$$

with optimal value $V_N^*(q)$. We suppose that this restriction can be solved much faster than P. If q is a binary vector and N is a matrix with components $n_{ij} \in \{0, \pm 1\}$ then the constraints $Ny \leq q$ will be of the form of fixing components of y or simple clique constraints or bounding constraints like $y_i \leq y_j$, all of which speed up a branch-and-bound algorithm. We define $R = \{r |\ r = Ny \text{ for some } (x, y) \in S_P\}$ to be the set of vectors associated with feasible solutions to problem P and for this chapter assume that N is chosen such that R contains only binary vectors.

Clearly, we have $V_P^* \geq V_N^*(r)\ \forall\ r \in R$ and $V_P^* = V_N^*(r^*)$ with $r^* = Ny_P^*$ for an optimal solution (x_P^*, y_P^*) to P. Thus, a strategy for finding an optimal solution to P is searching over the set R and solving restrictions $P_N(q)$, which will be feasible if there is an $r \in R$ such that $q \geq r$. A major advantage of such a strategy is that it produces a feasible solution to P each time such a restriction $P_N(q)$ is solved. If we restrict ourselves to solving restrictions $P_N(r)$ for vectors $r \in R$, then we can strengthen the restriction $P_N(r)$ by replacing the constraints $Ny \leq r$ with $Ny = r$. However, we will see in Section 3.2 that it can be beneficial to consider vectors q that are not necessarily in R and hence we use the formulation $P_N(q)$ given by (12).

Ideally, we would only solve restrictions $P_N(q)$ whose optimal value $V_N^*(q)$ is close to the optimal value V_P^*. Consequently, we would need an oracle that considers all vectors $r \in R$, but returns only those with $V_N^*(r) \approx V_P^*$. The role of this oracle is thus similar to the role of the pricing problem in column generation: consider all columns, but return only columns

with positive reduced costs (for a maximization problem).

Therefore, we next assume that we know the entire set R and build a model that extends the formulation of P to both choose a vector r from R and solve the resulting restriction $P_N(r)$. Specifically, we define the problem MP:

$$\begin{array}{llllll} \max & cx & + & dy & & \\ \text{s.t.} & Ax & + & By & & = b \\ & & & Ny & -Rz & \leq 0 \\ & & & & 1z & = 1 \\ & & & x \text{ real}, & y, z \text{ binary}, & \end{array} \tag{13}$$

where the binary variables z in MP represent the choice of vector r for which the restriction $P_N(r)$ should be solved. Let V^*_{MP} denote the optimal value of MP, MLP its linear relaxation and V^*_{MLP} its optimal value. In addition, by replacing $Ny - Rz \leq 0$ with $Ny - Rz = 0$, we obtain problem $MP_=$ with linear relaxation $MLP_=$ and optimal linear relaxation value $V^*_{MLP_=}$.

We first observe that MP and $MP_=$ are equivalent reformulations of P and that their linear relaxations are no weaker than LP.

Proposition 3.1.1 *$V^*_{MP_=} = V^*_{MP} = V^*_P$ and $V^*_{MLP_=} \leq V^*_{MLP} \leq V^*_{LP}$.*

Proof As MP contains the constraint set $Ny \leq Rz$ as well as the variables and constraints defining P we must have $V^*_{MP} \leq V^*_P$. Since $MP_=$ is defined with $Ny = Rz$ instead of $Ny \leq Rz$ we have $V^*_{MP_=} \leq V^*_{MP}$. Similar reasoning yields $V^*_{MLP_=} \leq V^*_{MLP} \leq V^*_{LP}$. However, for an optimal solution (x^*_P, y^*_P) to P and $r^* = Ny^*_P$ we know that $r^* \in R$ given the definition of R. Thus (x^*_P, y^*_P, e_{r^*}) is a feasible solution with objective function value V^*_P for both MP and $MP_=$ and we have $V^*_{MP_=} = V^*_{MP} = V^*_P$.

Although the primary purpose of MP and $MP_=$ is to aid the search for high-quality primal solutions, we next see that MLP may in fact provide a tighter bound on V^*_P than LP, i.e.

we can have $V^*_{LP} > V^*_{MLP}$. Consider the following integer program:

$$\begin{aligned} V^*_P = \text{maximize} \quad & -10y_3 + -y_4 \\ \text{s.t.} \quad & y_1 + y_2 = 1 \\ & 10y_1 \leq 10y_3 \\ & 10y_2 \leq 9y_4 \\ & y_1, y_2, y_3, y_4 \in \{0, 1\} \end{aligned}$$

and the extended formulation MP associated with the restriction created by adding to P the constraints $y_3 \leq r_3, y_4 \leq r_4$. The only feasible solution to P is $y_1 = y_3 = 1, y_2 = y_4 = 0$. Therefore, we have $V^*_P = -10$, the only element of R is $\bar{r}$ with $\bar{r}_3 = 1, \bar{r}_4 = 0$ and every (including the optimal) solution to MLP must have $z_{\bar{r}} = 1$. Thus, the constraint set $Ny - Rz \leq 0$ reduces to $y_3 \leq \bar{r}_3 = 1$, which is redundant, and $y_4 \leq \bar{r}_4 = 0$, which fixes the variable $y_4 = 0$. As a result, the optimal solution to MLP is integral, namely it is $y_1 = y_3 = 1$ and we have $V^*_{MLP} = -10$. On the other hand, the solution to LP is $y_2 = .9, y_4 = 1, y_1 = y_3 = .1$ and $V^*_{LP} = 1 * -1 + .1 * -10 = -2$.

Finally, we see that defining $MP_=$ with $Ny = Rz$ instead of $Ny \leq Rz$ may yield an even tighter linear relaxation, i.e. we can have $V^*_{MLP} > V^*_{MLP,=}$. Consider the following integer program:

$$\begin{aligned} V^*_P = \text{maximize} \quad & -10y_2 \\ \text{s.t.} \quad & y_1 = 1 \\ & y_1 \leq 10y_2 \\ & y_1, y_2 \in \{0, 1\}. \end{aligned}$$

and the extended formulations MP ($MP_=$) associated with the restriction created by adding to P the constraints $y_2 \leq r_2 (y_2 = r_2)$. Again, P has a single feasible solution, namely $y_1 = y_2 = 1$ with $V^*_P = -10$ and R consists of a single vector $\bar{r}$, with $\bar{r}_2 = 1$. Therefore, every (including the optimal) solution to MLP and $MLP_=$ must have $z_{\bar{r}} = 1$. For MLP, the constraint set $Ny - Rz \leq 0$ reduces to $y_2 \leq \bar{r}_{a_1} = 1$ which is redundant and we have $V^*_{MLP,\leq} = V^*_{LP} = -10 * .1$ since the optimal solution to LP is $y_1 = 1, y_2 = .1$. For $MLP_=$, the constraint set $Ny - Rz = 0$ reduces to $y_2 = \bar{r}_{a_1} = 1$ which is not redundant and induces the optimal solution to P.

Since instances of both MLP and $MLP_{=}$ have to be solved by column generation, the basis of our approach for solving instances of MP and $MP_{=}$ is a branch-and-price algorithm. We discuss the branch-and-price algorithm and procedures for speeding up the search for high-quality solutions in the next section. Although we can simply add steps to the branch-and-price scheme to perform these procedures, we leverage the fact that solving $P_N(q)$ for $q \geq r$ for some $r \in \bar{R}$ can be done independently from the execution of the branch-and-prie scheme and develop a more powerful framework specifically geared towards parallel computer architectures.

3.2 Solving Instance of $MP, MP_{=}$

We have presented two reformulations of P, MP and $MP_{=}$, each of which are candidates for a branch-and-price algorithm. While $MLP_{=}$ may provide a tighter bound on V_P^* than MLP it may be a more difficult linear program to solve. For example, we will see that when solving MLP for MCFCNF, the non-negative dual variables associated with $Ny - Rz \leq 0$ yield a relatively easy pricing problem, whereas when solving $MLP_{=}$ for MCFCNF, the unconstrained dual variables associated with $Ny - Rz = 0$ yield a relatively hard pricing problem. Thus we restrict the presentation of our approach to MP, although all the steps can be easily adapted to $MP_{=}$.

A branch-and-price algorithm consists of the following components: a restricted master problem, a pricing problem, and a branching scheme. We next illustrate how we define each of these components. Let $\bar{R} \subseteq R$ be a set of known restrictions of P. The restricted master problem RMP is defined as:

$$\begin{array}{lrcrcll} \max & cx & + & dy & & & \\ \text{s.t.} & Ax & + & By & & = b & \\ & & & Ny & -\bar{R}z & \leq 0 & (\pi) \\ & & & & 1z & = 1 & (\alpha) \end{array} \qquad (14)$$

$$x \text{ real}, \quad y, z \text{ binary}.$$

with dual variables $\pi \geq 0$ and α unrestricted. Let V_{RMP}^* denote the optimal value of RMP, $RMLP$ the linear relaxation of RMP and V_{RMLP}^* its optimal value.

To ensure that we solve MP and in turn find an optimal solution (x_P^*, y_P^*) to P, we need a mechanism for generating columns to add to $\bar{R}$ which will eventually produce $r^* = Ny_P^*$. By searching for a column $r \in R \setminus \bar{R}$ with positive reduced cost, i.e. with $\pi r - \alpha > 0$, the pricing problem $G(\pi, \alpha)$ associated with $RMLP$ will do exactly that. Specifically, $G(\pi, \alpha)$ is:

$$\begin{aligned} \max \quad & \pi r - \alpha \\ \text{s.t.} \quad & r \in R \\ & r \text{ binary.} \end{aligned} \tag{15}$$

Observe that since our pricing problem searches for variables that are not in the formulation of P, we can strengthen $RMLP$ with inequalities $fx + gy \leq \gamma$ that are valid for P without affecting the structure of the pricing problem. This column generation process also yields a dual bound. Specifically, if r^* represents the optimal solution to the pricing problem and $u \geq \pi r^* - \alpha$, then we have $V_{RMLP}^* + u \geq V_{MLP}^* \geq V_{MP}^* = V_P^*$.

The last component to define is a branching scheme to handle the situation where the column generation process has solved MLP but the optimal solution $(x_{RMLP}^*, y_{RMLP}^*, z_{RMLP}^*)$ to $RMLP$ is fractional. We consider two cases:

1. z_{RMLP}^* is binary but (x_{RMLP}^*, y_{RMLP}^*) is fractional. For this case we create the branches $z_r = 0$ and $z_r = 1$ for the vector r for which $z_{r,RMLP}^* = 1$. However, we immediately evaluate and prune the branch with $z_r = 1$ by solving $P_N(r)$. Note that branching on the z variable is equivalent to removing columns from $\bar{R}$ so that if we have already solved $P_N(r_1)$ we can add constraints to (15) to exclude r_1. In fact, since we know that $r_2 \leq r_1 \Rightarrow V_N^*(r_2) \leq V_N^*(r_1)$, we add constraints to exclude both r_1 and elements from R whose restrictions cannot yield a better solution than the optimal solution to $P_N(r_1)$. In particular, we define $R_D = \{r \in R \parallel P_N(r) \text{ already solved } \}$ and for each $r^i \in R_D$ we add to (15) the constraint $\sum_{j:r_j^i=0} r_j \geq 1$ since we are only interested in vectors r that contain an element $j \notin r^i$.

2. z_{RMLP}^* is fractional. For this case, we choose a row i with $0 < (\bar{R}z_{RMLP}^*)_i < 1$ and enforce $(Rz)_i \leq 0$ on one branch and $(Rz)_i \geq 1$ on the other. Note that we can satisfy $(Rz)_i \leq 0 \ \ (\geq 1)$ in the pricing problem simply by setting $r_i = 0 \ \ (= 1)$.

Algorithm 7 Branch-and-price (BP)

set $\bar{R} = \emptyset$, $R_D = \emptyset$
while MP has not been solved **do**
  select an unevaluated node
  while $RMLP$ has not been solved **do**
    solve $RMLP$ only considering columns of $\bar{R} \setminus R_D$ valid for this node
    solve the pricing problem to find an improving vector r to add to $\bar{R}$
  end while
  branching if necessary
end while

We summarize the branch-and-price scheme (BP) in Algorithm 7. This branch-and-price scheme will, when given sufficient time, produce an optimal solution. However, its most important feature is that it automatically generates restrictions $r \in R$ for which $P_N(r)$ may produce a high-quality solution to P. We next present the Primal Solution Construction (PC) algorithm defined in Algorithm 8 and the Primal Solution Improvement (PI) local search heuristic defined in Algorithm 9 to show how this feature can be exploited effectively to generate high-quality feasible solutions .

Given a set of restriction vectors $\bar{R}^{PC}$, PC produces feasible solutions by choosing an $r \in \bar{R}^{PC}$ and solving $P_N(r)$. When PC is given the value v^{BEST} of the current best feasible solution, we strengthen $P_N(r)$ by adding the constraint $cx+dy \geq v^{BEST}$ to reflect that only primal solutions with objective function value higher than the current best are of interest.

Algorithm 8 *Primal Solution Construction (PC)*

Require: $\bar{R}^{PC}, R_D^{PC}, (x^*_{RMLP}, y^*_{RMLP}, z^*_{RMLP})$ and best-known primal solution value v^{BEST} if exists
if $\exists\, r \in \bar{R}^{PC} \setminus R_D^{PC}$ such that $z^*_{r,RMLP} > 0$ **then**
  set r to such an r with the highest value of $z^*_{r,RMLP}$
else if $\bar{R}^{PC} \setminus R_D^{PC} \neq \emptyset$ **then**
  set r to a randomly chosen vector from $\bar{R}^{PC} \setminus R_D^{PC}$
else
  convert $\tilde{r} = \bar{R} z^*_{RMLP}$ to an integral vector and set $r = \tilde{r}$
end if
Solve $P_N(r)$ with the constraint $cx + dy \geq v^{BEST}$ and add r to R_D^{PC}
return R_D^{PC} and new best solution if found

We can also create an $\tilde{r}$ such that solving $P_N(\tilde{r})$ represents searching a neighborhood of the current best solution (x_P^{BEST}, y_P^{BEST}). Observe that $\tilde{r} \geq r^{BEST} = N y_P^{BEST}$ implies

$V_N^*(\tilde{r}) \geq V_N^*(r^{BEST})$. This suggests solving $P_N(\tilde{r})$ with an $\tilde{r} > r^{BEST}$. The PI algorithm does so, and uses a variety of schemes for constructing $\tilde{r}$. We choose which scheme to use in each call to PI with a method similar to the biased roulette wheel approach used in Chapter 2.

Scheme augment-best: To obtain a $\tilde{r} > r^{BEST}$, we start from $\tilde{r} = r^{BEST}$ and then choose a subset of indices i with $r_i^{BEST} = 0$ and set $\tilde{r}_i = 1$. We use two metrics, both based on the solution to $RMLP$, to choose the indices i for which we set $\tilde{r}_i = 1$. The first metric is the dual variable π_i associated with $(Ny - \bar{R}z \leq 0)_i$ and the second is the value $(\bar{R}^{PI} z^*_{RMLP})_i$. The higher these values are, the more likely it is that we choose index i. We refer to solutions created in this way as belonging to the *augment-best* neighborhood.

The next two schemes use the concept of path-relinking, i.e. by combining the structural information from two good solutions we may produce a restriction that yields an even better solution.

Scheme best-relink-other-r: The first path-relinking mechanism uses vectors generated by the column generation process. Specifically, we choose a random $r \in \bar{R}^{PI}$ and set $\tilde{r}_i = max(r_i, r_i^{BEST})$. We refer to this neighborhood as the *best-relink-other-r* neighborhood.

Scheme best-relink-other-solution: The next path-relinking mechanism uses vectors induced by other solutions. Specifically, we set $\tilde{r}_i = max(r_i, r_i^{BEST})$ where $r = Ny_P^1$ for a solution (x_P^1, y_P^1) that was found either earlier in the execution of the current process or potentially by another process. We refer to this neighborhood as the *best-relink-other-solution* neighborhood.

With these local search ideas care has to be taken that we do not create a vector $\tilde{r}$ with too many indices i with $\tilde{r}_i = 1$, as the resulting restriction may be too loosely constrained to be solved quickly. In our current implementation we ensure that all $\tilde{r}$ satisfy $\|\tilde{r}\|_1 \leq \gamma$ where γ is an algorithm parameter.

Algorithm 9 *Primal Solution Improvement (PI)*

Require: $\bar{R}^{PI}, (x^*_{RMLP}, y^*_{RMLP}, z^*_{RMLP}), \pi, (x_P^{BEST}, y_P^{BEST})$
  Choose scheme for creating $\tilde{r}$
  Solve $P_N(\tilde{r})$
  return new solution if found

As noted, instead of inserting calls to PC and PI into the BP algorithm, we execute them in parallel and define Branch-and-Price Guided Search (BPGS) as the parallel execution of BP (Algorithm 7), PC (Algorithm 8) and PI (Algorithm 9). By executing PC in parallel to BP we can solve $P_N(r)$ for r's created by the column generation process and price out new r's at the same time. We present in Algorithms 10,11 and 12 the parallelized versions of BP (PBP),PC (PPC) and PI (PPI). We note that while we include steps in these algorithms for broadcasting data to and checking for new data from other processes, in our current implementation these steps do not necessarily occur at every iteration

Algorithm 10 Parallelized Branch-and-price (PBP)

set $\bar{R}^{BP} = \emptyset$,$R_D^{BP} = \emptyset$
while MP has not been solved **do**
  select an unevaluated node
  while $RMLP$ has not been solved **do**
    solve $RMLP$ only considering columns of $\bar{R}^{BP} \setminus R_D^{BP}$ valid for this node
    solve the pricing problem to find an improving vector r to add to $\bar{R}^{BP}$
    Broadcast $\bar{R}^{BP} \setminus R_D^{BP}, (x^*_{RMLP}, y^*_{RMLP}, z^*_{RMLP})$ to PPC and PPI processes
    Broadcast π to PPI processes
    Check for set R_D^{PC} from PPC process and set $R_D^{BP} = R_D^{BP} \cup R_D^{PC}$ if found
    Check for new solution from PPC,PPI processes
  end while
  branching if necessary
end while

Algorithm 11 *Parallelized Primal Solution Construction (PPC)*

$\bar{R}^{PC} = \emptyset$,$R_D^{PC} = \emptyset$
while Not told to stop **do**
  check for new $\bar{R}^{BP}, (x^*_{RMLP}, y^*_{RMLP}, z^*_{RMLP})$ from PBP process and set $\bar{R}^{PC} = \bar{R}^{PC} \cup \bar{R}^{BP} \setminus R_D^{PC}$ if new $\bar{R}^{BP}$ found
  check for new best solution value v^{BEST} from all processes if exists
  call PC with $\bar{R}^{PC}, R_D^{PC}, (x^*_{RMLP}, y^*_{RMLP}, z^*_{RMLP}), v^{BEST}$
  broadcast R_D^{PC} to PBP process and new solution if found to all processes
end while

A natural parallelization scheme is to dedicate a single process to the execution of PBP, m_1 processes to the execution of PPC and m_2 processes to the execution of PPI. The key information that needs to be passed between the processes is the set of restrictions $\bar{R}$, those which have been solved, $\bar{R}_D$, and the current best solution (x_P^{BEST}, y_P^{BEST}), which can be done by passing $r^{BEST} = Ny_P^{BEST}$. In the application discussed below, passing the set of

Algorithm 12 *Parallelized Primal Solution Improvement (PPI)*

$\bar{R}^{PI} = \emptyset$
while Not told to stop **do**
  check for new $\bar{R}^{BP}, (x^*_{RMLP}, y^*_{RMLP}, z^*_{RMLP}), \pi$ from PBP process, set $\bar{R}^{PI} = \bar{R}^{BP}$ if found
  check for new best solution (x_P^{BEST}, y_P^{BEST}) from all processes
  call PI with $\bar{R}^{PI}, (x^*_{RMLP}, y^*_{RMLP}, z^*_{RMLP}), \pi$ and (x_P^{BEST}, y_P^{BEST})
  broadcast new solution if found
end while

restrictions $\bar{R}$ is easily manageable and does not lead to communication delays because the set does not get too large during the execution of the algorithm and a restriction, which is defined by a 0-1 vector, can be encoded compactly. In addition, while conceptually the sets R_D are passed between processes, it is unnecessary to pass the actual r's. Specifically, PBP assigns a unique identifier to each $r \in \bar{R}^{BP}$ which is also shared with other processes. This allows PPC to pass just the identifiers of the elements in R_D^{PC}. Since PPI generates it's own $\tilde{r}$ which are unknown to PBP we chose to not pass them to avoid potential communication delays. We summarize the passing of data between processes in Figure 1. To save space

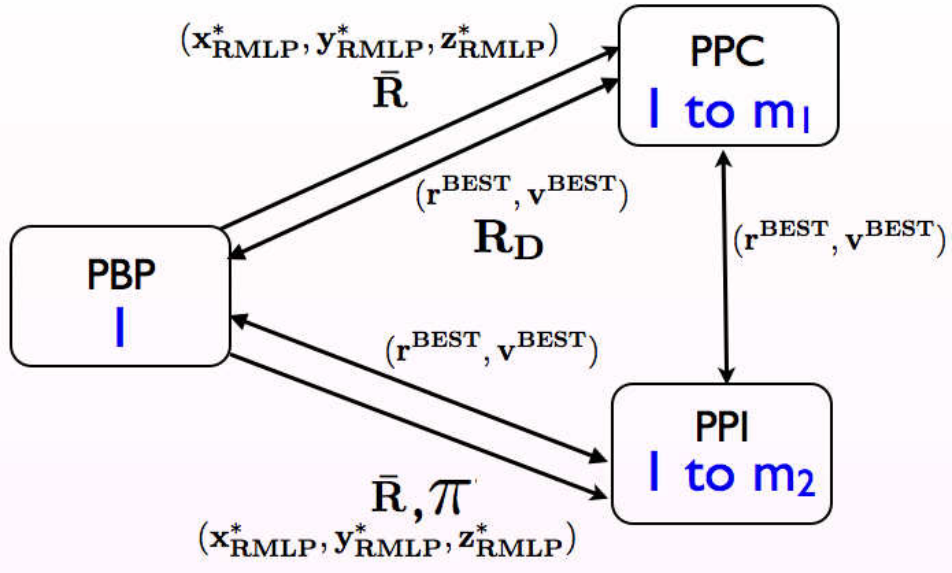

Figure 1: Data sharing between processes

we omit superscripting the sets $\bar{R}, R_D$ with their process owner. Although not depicted in the figure, we also pass solutions between PPI processes to facilitate path-relinking in the *best-relink-other-solution* neighborhood. Finally, we note that if a PPC process runs out of

restrictions to solve, then instead of idling until it receives a new $\bar{R}^{BP}$ it can temporarily change its mode to that of a PPI process.

3.3 Multi-Commodity Fixed-Charge Network Flow

We next discuss how BPGS can be used to solve instances of the single-path variant of MCFCNF, the formulation of which is presented in Section 2.2. In keeping with the approach presented in Sections 3.1 and 3.2, we use the equivalent objective function

$$\max \sum_{k \in K} \sum_{(i,j) \in A} -c_{ij}(d^k x_{ij}^k) + \sum_{(i,j) \in A} -f_{ij} y_{ij}$$

The BPGS approach presented in Section 3.2 has many generic tasks. In particular, once we have chosen a matrix N to dictate the structure of the restriction $P_N(r)$, the algorithms PC and PI require no further specialization. To apply the approach to MCFCNF, we chose the matrix $N = I$ to indicate how the approach can generalize "variable fixing" heuristics. Hence, $P_N(r)$ consists of the extra constraints $y_{ij} \leq r_{ij} \ \ \forall (i,j) \in A$. Thus, we are in essence searching the space of feasible network designs for the optimal solution to P.

Given the dual variables π_{ij}, the pricing problem is

$$\max \sum_{(i,j) \in A} \pi_{ij} r_{ij} - \alpha$$

subject to

$$\sum_{j:(i,j) \in A} x_{ij}^k - \sum_{j:(j,i) \in A} x_{ji}^k = \delta_i^k \quad \forall i \in N, \ \forall k \in K, \tag{16}$$

$$\sum_{k \in K} d^k x_{ij}^k \leq u_{ij} r_{ij} \ \ \forall (i,j) \in A, \tag{17}$$

$$r_{ij} \in \{0,1\} \ \ \forall (i,j) \in A, \tag{18}$$

$$x_{ij}^k \in \{0,1\} \quad \forall k \in K, \ \forall (i,j) \in A. \tag{19}$$

Note that while the feasible region of the pricing problem is the same as that of the original problem, the objective function is significantly different and makes the problem easier to solve than the original. Since it is a maximization problem and the dual variables

π are nonnegative, the objective function encourages the variables r_{ij} to be set to 1 even in solutions to the linear programming relaxation (LPR), thus hopefully avoiding highly fractional solutions. However, there is also an optimal solution with $r_{ij} = 1\ \forall (i,j) \in A$. This is not desirable since the restricted problem associated with this r will be too loosely constrained to be solvable. To eliminate this problem we use the structure of an optimal solution to MCFCNF. In particular, given our assumption that $f_{ij} \geq 0\ \forall (i,j) \in A$, we can assume that in an optimal solution an arc is installed only if it is used by a commodity. Thus we add the inequalities

$$r_{ij} \leq \sum_{k \in K} x_{ij}^k \quad \forall (i,j) \in A \tag{20}$$

to the pricing problem. We also observe that $c_{ij} \geq 0\ \forall (i,j) \in A$ implies that the route a commodity takes from its source node to its sink node will not contain any cycles. Thus, to reduce circuity in the routes chosen for commodities by the pricing problem, we add the constraints

$$\sum_{j:(i,j) \in A} x_{ij}^k \leq 1 \quad \forall j \in N,\ \forall k \in K, \tag{21}$$

$$\sum_{i:(i,src(k)) \in A} x_{ij}^k \leq 0 \quad \forall k \in K, \tag{22}$$

and

$$\sum_{j:(sink(k),j) \in A} x_{ij}^k \leq 0 \quad \forall k \in K. \tag{23}$$

The first set ensures that a commodity does not leave a node multiple times, whereas the next two sets ensure that a commodity never enters (leaves) its source (sink).

Finally, we have chosen to try to strengthen the bound V_{MLP}^* through the use of well-known valid inequalities for MCFCNF. Specifically, we augmented BP to a branch-price-and-cut algorithm, adding steps that dynamically add the disaggregated constraints (6) and the *cover* inequalities $\sum_{k \in C} x_{ij}^k \leq |C| - 1$ to $RMLP$ where a cover with respect to an arc (i,j) is a set $C \subseteq K$ such that $\sum_{k \in C} d^k > u_{ij}$.

3.4 Computational Results

There are many questions to answer regarding the performance of BPGS. Since it is an exact algorithm, we chose to benchmark it against a commercial MIP solver: CPLEX 11.2.

This allows us to assess the capability of BPGS to produce high quality primal solutions, tight dual bounds, and proofs of optimality. BPGS has been implemented in C++ and uses CPLEX as its MIP/LP solver. All experiments were performed on a machine with 8 Intel Xeon CPUs running at 2.66 GHz and with 32 GB RAM. The basic parallelization consisted of 1 PBP process, 1 PPC process and 2 PPI processes. Message passing between processes was implemented via a combination of MPI ([17]) and text files. The use of 4 processors seems reasonable as many high-end desktops come with 4 processors. Finally, for all experiments we allowed our code to run for 15 minutes and all restrictions $P_N(r)$ were solved with an optimality tolerance set to 1% and a time limit of 5 minutes.

In our first set of experiments, we ran CPLEX for 6 hours. Since we want to assess both primal- and dual-side performance of BPGS, CPLEX was run with MIPEmphasis set to *balanced*, which means that CPLEX "uses tactics intended to find a proven optimal solution quickly" ([20]). The disaggregate constraints $x_{ij}^k \leq y_{ij}$ are implemented as CPLEX *user cuts*, for the CPLEX experiments and when branch-and-price guided search solves restrictions $P_N(r)$. Unless otherwise noted, computation times are reported in seconds.

Of the three sets of MCFCNF instances presented in [16] and identified as C, C+ and R, we chose to use the instances in C due to their diversity in size and difficulty. The instances have been used to benchmark the performance of meta-heuristics designed for the MCFCNF variant in which a commodity may be split across multiple paths ($0 \leq x_{ij}^k \leq 1$). However, the instances remain feasible when splitting is not allowed. The instances are classified as follows: F denotes instances where the ratio of fixed to variable cost for an arc is high and V otherwise, T denotes instances that are tightly capacitated relative to total demand and L otherwise. Finally, the naming scheme is $\#nodes - \#arcs - \#commodities - (F|V) - (T|L)$.

Unfortunately, we do not have access to the parallel version of CPLEX 11.2. However, the results clearly show that BPGS produces near-optimal solutions quickly. Furthermore, the frequency with which BPGS produces a better solution in 15 minutes than CPLEX in 3 hours suggests that it is likely to be competitive with a parallel version of CPLEX.

3.4.1 Primal Side

Table 6 shows the results for the 31 instances considered. (Note that we have used the minimization form of MCFCNF.) More specifically, we give

- the value of the best primal solution found by CPLEX in 6 hours (CPLEX Primal) and the time taken by CPLEX to find that solution (CPLEX TTB),
- the value of the best primal solution found by BPGS in 15 minutes (BPGS Primal) and the time taken by BPGS to find that solution (BPGS TTB),
- a comparison of the quality of primal solutions produced by BPGS and CPLEX in the form of $100 \times$ (BPGS Primal $-$ CPLEX Primal)/(BPGS Primal),
- in case CPLEX finds a better solution than BPGS, the time required to find the first better solution (Time to Beat),
- the value of the dual bound found by CPLEX (CPLEX Dual) and the resulting optimality gap $100 \times$ (CPLEX Primal $-$ CPLEX Dual)/(CPLEX Primal) (CPLEX Gap),
- the value of the dual bound found by BPGS (BPGS Dual) and the resulting optimality gap $100 \times$ (BPGS Primal $-$ BPGS Dual)/(BPGS Primal) (BPGS Gap),
- the quality of the solution found by BPGS measured against the best dual bound $100 \times$(BPGS Primal$-$max(CPLEX Dual, BPGS Dual))/(BPGS Primal) (BPGS Best Gap),
- a comparison of the strength of the dual bounds produced by the BPGS and CPLEX in the form of $100 \times$ (CPLEX Dual $-$ BPGS Dual)/(CPLEX Dual) (Dual Gap),
- the time CPLEX needed to prove the optimality of a solution when it was able to do so (Time to Opt CPLEX), and
- the time BPGS needed to prove the optimality of a solution when it was able to do so (Time to Opt BPGS).

We see that BPGS produces high-quality solutions quickly since it produces primal solutions in 15 minutes that are on average within 2.16% of optimality (only 0.02% worse than CPLEX). We observe that CPLEX also performs quite well, producing primal solutions in 6 hours that are on average within 2.14% of optimality. Furthermore, for 22 of the 31 instances BPGS produces a better primal solution. However, we note that we are counting instances where CPLEX quickly produces a solution whose value is within 1% of the optimal value, yet is worse than the value of the solution produced by BPGS (e.g. instance 20-230-40-F-T). For these instances, CPLEX may have found an even better solution if given a smaller optimality tolerance. Therefore, we next focus on the 21 instances (presented in boldface) for which CPLEX produced a solution within 1% of the optimal value that is better than the solution produced by BPGS or did not find a provably optimal solution at all. For 10 of these 21 instances BPGS produces a better solution in 15 minutes than CPLEX produces in 6 hours. For 10 of the remaining 11 instances, CPLEX requires nearly 3 hours (on average) to produce a solution that is better than what BPGS produces in 15 minutes (and those solutions are only 0.53% better on average).

Furthermore, we observe that for three of the instances, namely 20-300-200-F-L, 20-300-200-V-T, and 30-520-400-V-L, the best solution found by BPGS was produced by the PPC process whereas the best solutions for all other instances were produced by a PPI process. Thus local improvement seems to be quite effective. As each PPI process uses three different schemes for creating restrictions that represent a neighborhood of the best-known solution, we next analyze the performance of each. The *augment-best* neighborhood was responsible for the majority of the solution improvement over all instances at 86% followed by the *best-relink-other-r* neighborhood at 11% and the *best-relink-other-solution* neighborhood at 3%. One reason that the *best-relink-other-solution* neighborhood is not that effective may be the lack of intelligence in the selection of the solution to "link" it with. Currently, it selects the other solution randomly without considering how similar of dissimilar it is.

Table 6: Primal and Dual Comparison with default CPLEX

	CPLEX		BPGS		Gap with	Time	CPLEX		BPGS		BPGS Gap	Dual	Time to Opt	
Instance	Primal	TTB	Primal	TTB	CPLEX	To Beat	Dual	Gap	Dual	Gap	Best Dual	Gap	CPLEX	BPGS
20-230-40-V-L	423,933	1	423,933	15	0.00		423,830	0.02	419,868	0.96	0.02	0.93	1	55
20-230-40-V-T	399,148	1	398,870	16	-0.07		396,606	0.64	370,221	7.18	0.57	6.65	1	
20-230-40-F-T	669,587	1	668,699	78	-0.13		668,645	0.14	637,279	4.70	0.01	4.69	1	
20-230-200-V-L	**95,273**	**20,082**	**94,820**	**621**	**-0.48**		**93,373**	**1.99**	**84,392**	**11.00**	**1.53**	**9.62**		
20-230-200-F-L	**138,287**	**15,247**	**138,830**	**741**	**0.39**	**13,353**	**134,808**	**2.52**	**118,050**	**14.97**	**2.90**	**12.43**		
20-230-200-V-T	**99,074**	**19,599**	**98,999**	**625**	**-0.08**		**97,371**	**1.72**	**90,013**	**9.08**	**1.64**	**7.56**		
20-230-200-F-T	**139,898**	**20,718**	**139,718**	**818**	**-0.13**		**133,398**	**4.65**	**121,707**	**12.89**	**4.52**	**8.76**		
20-300-40-V-L	433,320	1	430,253	87	-0.71		429,963	0.77	428,013	0.52	0.07	0.45	1	153
20-300-40-F-L	599,917	1	597,059	32	-0.48		595,069	0.81	592,377	0.78	0.33	0.45	1	210
20-300-40-V-T	502,512	1	501,766	173	-0.15		500,409	0.42	470,201	6.29	0.27	6.04	1	
20-300-40-F-T	648,314	1	643,395	122	-0.76		642,075	0.96	604,682	6.02	0.21	5.82	1	
20-230-200-V-L	**76,392**	**12,915**	**76,389**	**447**	**-0.00**		**74,142**	**2.95**	**70,046**	**8.30**	**2.94**	**5.52**		
20-300-200-F-L	**119,304**	**21,144**	**118,756**	**865**	**-0.46**		**112,374**	**5.81**	**105,131**	**11.47**	**5.37**	**6.45**		
20-300-200-V-T	**76,486**	**18,401**	**76,501**	**823**	**0.02**	**18,401**	**75,221**	**1.65**	**71,841**	**6.09**	**1.67**	**4.49**		
20-300-200-F-T	**110,417**	**20,386**	**110,392**	**594**	**-0.02**		**105,116**	**4.80**	**98,684**	**10.61**	**4.78**	**6.12**		
30-520-100-V-L	**54,466**	**62**	**54,533**	**431**	**0.12**	**62**	**53,919**	**1.00**	**51,598**	**5.38**	**1.13**	**4.30**	**79**	
30-520-100-F-L	**96,348**	**7,766**	**96,311**	**813**	**-0.04**		**93,218**	**3.25**	**87,023**	**9.64**	**3.21**	**6.65**		
30-520-100-V-T	53,957	33	53,928	110	-0.05		53,509	0.83	46,312	14.12	0.78	13.45	33	
30-520-100-F-T	**99,396**	**4,952**	**99,391**	**806**	**-0.01**		**97,883**	**1.52**	**87,396**	**12.07**	**1.52**	**10.71**		
30-520-400-V-L	**114,902**	**20,090**	**114,949**	**880**	**0.04**	**19,900**	**112,045**	**2.49**	**102,044**	**11.23**	**2.53**	**8.93**		
30-520-400-F-L	**152,026**	**21,288**	**152,863**	**859**	**0.55**	**5,625**	**147,189**	**3.18**	**131,073**	**14.25**	**3.71**	**10.95**		
30-520-400-V-T	**116,485**	**21,107**	**116,826**	**873**	**0.29**	**10,724**	**114,647**	**1.58**	**104,040**	**10.94**	**1.87**	**9.25**		
30-520-400-F-T	**156,660**	**21,459**	**157,929**	**894**	**0.80**	**18,587**	**150,341**	**4.03**	**136,023**	**13.87**	**4.80**	**9.52**		
30-700-100-V-L	48,064	11	47,883	435	-0.38		47,585	1.00	46,332	3.24	0.62	2.63	11	
30-700-100-F-L	**60,384**	**12,488**	**60,805**	**682**	**0.69**	**8,849**	**59,869**	**0.85**	**56,091**	**7.75**	**1.54**	**6.31**	**12,488**	
30-700-100-V-T	47,781	22	47,670	217	-0.23		47,283	1.04	43,977	7.75	0.81	6.99	51	
30-700-100-F-T	**56,804**	**1,330**	**56,804**	**440**	**0.00**		**56,274**	**0.93**	**51,100**	**10.04**	**0.93**	**9.19**	**1,330**	
30-700-400-V-L	**99,087**	**20,643**	**98,949**	**722**	**-0.14**		**96,876**	**2.23**	**84,346**	**14.76**	**2.10**	**12.93**		
30-700-400-F-L	**139,839**	**21,307**	**139,375**	**886**	**-0.33**		**130,984**	**6.33**	**114,326**	**17.97**	**6.02**	**12.72**		
30-700-400-V-T	**97,092**	**18,122**	**98,260**	**800**	**1.19**	**3,639**	**94,432**	**2.74**	**84,662**	**13.84**	**3.90**	**10.35**		
30-700-400-F-T	**132,820**	**20,491**	**134,464**	**814**	**1.22**	**6,742**	**128,027**	**3.61**	**113,731**	**15.42**	**4.79**	**11.17**		
Average					0.02	10,588		2.14		9.46	2.16	7.49	1,076	139

A closer examination of the instances for which CPLEX finds a better solution than BPGS, presented in Table 7, reveals that this happens most often for instances with 400 commodities. This may indicate that our choice of $N = I$ to define restrictions is not appropriate for instances with a large number of commodities. Even if $P_N(r)$ only considers 200 arcs, with 400 commodities the IP still has 80,000 binary variables and may be too large to solve quickly.

Table 7: Results by Number Commodities in an Instance

# Commodities	# Instance	# Instances CPLEX Found Better	Avg. Time CPLEX Needed to Find Better
100	5	2	4,455
200	8	2	15,877
400	8	6	10,869

The "time to best" information in Table 6 only provides partial insight on performance. To supplement this information, we present in Figure 2 more detailed information concerning the progression of the quality of the solution produced by BPGS over the course of its execution for the four instances with 20 nodes, 230 arcs and 200 commodities. At each point in time, we show the optimality gap for the best solution produced up to that point (as measured against the dual bound produced by CPLEX after 6 hours). We see that for each of the four instances BPGS has produced a solution that is within 5% of optimality in less than eight minutes. Thus, the "time to best" somewhat inflates the time required for BPGS to produce a solution of high quality. For example, for instance 20-230-200-F-T, BPGS finds its best solution (of value 139,718) after 818 seconds. However, the previous best solution (found after 416 seconds) was of value 139,719 and both are better than what CPLEX produces in 6 hours.

Given the similarities in both nature and goal of BPGS and local branching, we compare BPGS with CPLEX' implementation of local branching. In Table 8, we give the value of the best solution produced by BPGS in 15 minutes and the the value of the best solution produced by CPLEX' local branching in 1 hour. We focus our analysis on the difficult instances presented in boldface in Table 6. For 14 of the 21 instances BPGS finds a better solution in 15 minutes than CPLEX with local branching finds in 1 hour. Enabling local branching allows CPLEX to find a better solution than BPGS for only one new instance:

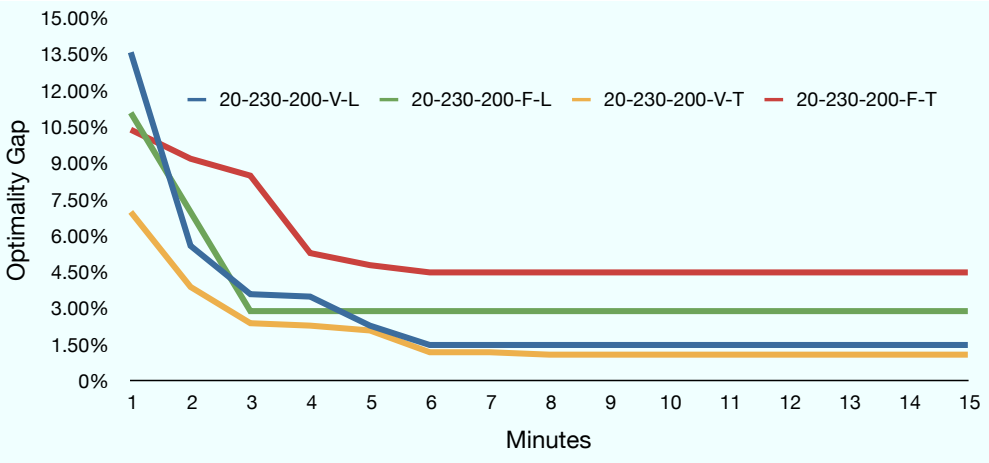

Figure 2: Primal-side Performance Over Time

30-700-100-F-T. We conclude from these experiments that BPGS is quite competitive with the implementation of local branching in CPLEX.

Next we study whether increasing the number of processes used by BPGS enables it to find good solutions even more quickly. Thus we compare our original parallelization that consisted of 4 processes broken down into 1 PBP process, 1 PPC process and 2 PPI processes with 6 processes broken down into 1 PBP process, 1 PPC process and 4 PPI processes. In Table 9 we present the results of this set of experiments. For each instance we present in boldface the best solution when there is not a tie or the lower time to best solution when there is. We note that for 21 of the instances using 6 processors either enabled the approach to find a better solution or an equivalent solution in less time than when we used 4 processors. We also note that for some instances (such as 20-230-200-F-T or 30-700-100-V-L) the time taken to find the best solution is decreased dramatically when we allot 6 processors to the approach.

3.4.2 Dual Side

To assess the dual-side performance of BPGS, we look again at the results reported in Table 6. We observe that while the dual bound produced by BPGS in 15 minutes is worse than the bound produced by CPLEX after 6 hours it is only 7.5% worse on average. Furthermore, at the end of 15 minutes BPGS produced a primal solution, a dual bound, and a provable optimality gap that was only 9.5% on average.

Table 8: Primal Comparison with CPLEX and Local Branching

	CPLEX + LB		BPGS		Gap with	Time
Instance	Primal	TTB	Primal	TTB	CPLEX + LB	To Beat
20-230-200-V-L	95,046	3,118	94,820	621	-0.24	
20-230-200-F-L	141,844	3,600	138,830	741	-2.17	
20-230-200-V-T	99,508	3,600	98,999	625	-0.51	
20-230-200-F-T	139,625	2,779	139,718	818	0.07	2,779
20-230-200-V-L	76,572	3,423	76,389	447	-0.24	
20-300-200-F-L	119,611	3,494	118,756	865	-0.72	
20-300-200-V-T	76,816	3,600	76,501	823	-0.41	
20-300-200-F-T	111,522	2,680	110,392	594	-1.02	
30-520-100-V-L	54,393	226	54,533	431	0.26	62
30-520-100-F-L	96,697	1,363	96,311	813	-0.40	
30-520-100-F-T	99,386	1,835	99,391	806	0.01	1,835
30-520-400-V-L	115,109	3,600	114,949	880	-0.14	
30-520-400-F-L	153,541	3,600	152,863	859	-0.44	
30-520-400-V-T	116,664	2,869	116,826	873	0.14	1,200
30-520-400-F-T	161,437	3,300	157,929	894	-2.22	
30-700-100-F-L	60,640	3,266	60,805	682	0.27	2,756
30-700-100-F-T	56,686	985	56,804	440	0.21	794
30-700-400-V-L	100,660	2,576	98,949	722	-1.73	
30-700-400-F-L	143,044	3,600	139,375	886	-2.63	
30-700-400-V-T	97,713	2,788	98,260	800	0.56	2,200
30-700-400-F-T	137,959	3,600	134,464	814	-2.60	
Average					-0.45	1,463

Examining the 13 instances that CPLEX solves to optimality, we see that 8 of those instances are solved at the root node through a combination of primal heuristics and valid inequalities added to strengthen the dual bound. Our implementation of BPGS does not have such enhancements, but is still able to solve 3 of them and produce primal solutions within 1% of the dual bound produced by CPLEX for the others, which implies that BPGS is unable to prove optimality only because its own dual bound is not strong enough.

3.4.3 Comparison with IP Search

Lastly we compare in Table 10 the performance of BPGS with the IP Search procedure for the MCFCNF presented in the previous chapter. We note that the comparison is not completely fair as the results for BPGS were achieved with the use of CPLEX 11.2 while the IP Search results were achieved with CPLEX 9. For each instance we present in boldface the best solution when there is not a tie or the lower time to best solution when there is. We see that for 23 instances BPGS either finds the best solution or a solution equivalent to the one found by IP Search in less time. Although the IP Search procedure was designed specifically for the MCFCNF, the solution it produced was 1% worse on average than the one produced by the more general BPGS. What is most striking is how much stronger the

Table 9: Primal Side Comparison of 4 and 6 Processors

	4 Processors		6 Processors	
Instance	Primal	TTB	Primal	TTB
20-230-40-V-L	423,933	**15**	423,933	44
20-230-40-V-T	398,870	16	398,870	**14**
20-230-40-F-T	668,699	78	668,699	**13**
20-230-200-V-L	94,820	621	94,820	**377**
20-230-200-F-L	138,830	741	**138,553**	360
20-230-200-V-T	**98,999**	625	99,009	356
20-230-200-F-T	139,718	818	**139,611**	491
20-300-40-V-L	430,253	87	430,253	**9**
20-300-40-F-L	597,059	32	597,059	**25**
20-300-40-V-T	501,766	173	501,766	**75**
20-300-40-F-T	643,395	122	643,395	**46**
20-230-200-V-L	76,389	447	**76,054**	865
20-300-200-F-L	118,756	865	**118,644**	384
20-300-200-V-T	**76,501**	823	76,670	751
20-300-200-F-T	110,392	594	**110,004**	880
30-520-100-V-L	54,533	431	**54,427**	686
30-520-100-F-L	**96,311**	813	96,342	645
30-520-100-V-T	53,928	110	**53,812**	814
30-520-100-F-T	**99,391**	806	99,429	808
30-520-400-V-L	**114,949**	880	115,708	720
30-520-400-F-L	152,863	859	**152,797**	850
30-520-400-V-T	**116,826**	873	116,827	800
30-520-400-F-T	157,929	894	**157,425**	823
30-700-100-V-L	47,883	435	47,883	**164**
30-700-100-F-L	**60,805**	682	60,818	190
30-700-100-V-T	47,670	**217**	47,670	660
30-700-100-F-T	56,804	440	**56,686**	573
30-700-400-V-L	**98,949**	722	99,292	727
30-700-400-F-L	139,375	886	**139,174**	880
30-700-400-V-T	98,260	800	**97,613**	872
30-700-400-F-T	134,464	814	**134,399**	870

dual bound produced by BPGS is than the dual bound produced by IP Search. Given that for these instances the extended formulation used by BPGS did not provide a stronger linear relaxation than the compact one, this is most likely due to the fact that BPGS performs its primal and dual side tasks in parallel whereas for IP Search these tasks compete for CPU time.

Having studied the potential of IP-based search on an academic network design problem, the MCFCNF, we next turn our attention to a large-scale network design problem of real-world interest. Specifically, we study the applicability of IP-based search concepts to the service network design problem for Less-Than-Truckload freight transportation carriers.

Table 10: Primal and Dual Comparison with IP Search

	BPGS		IP Search		Gap with	BPGS		IP Search	
Instance	Primal	TTB	Primal	TTB	BPGS	Dual	Gap	Dual	Gap
20-230-40-V-L	**423,933**	15	424,671	33	0.17	419,868	0.96	399,874	5.84
20-230-40-V-T	398,870	**16**	398,870	20	0.00	370,221	7.18	323,659	18.86
20-230-40-F-T	668,699	78	668,699	**35**	0.00	637,279	4.70	618,883	7.45
20-230-200-V-L	**94,820**	621	96,456	479	1.70	84,392	11.00	71,295	26.09
20-230-200-F-L	**138,830**	741	141,700	422	2.03	118,050	14.97	102,808	27.45
20-230-200-V-T	**98,999**	625	100,884	630	1.87	90,013	9.08	78,952	21.74
20-230-200-F-T	**139,718**	818	141,734	504	1.42	121,707	12.89	100,626	29.00
20-300-40-V-L	430,253	87	430,253	**5**	0.00	428,013	0.52	376,500	12.49
20-300-40-F-L	597,059	32	597,059	**19**	0.00	592,377	0.78	553,120	7.36
20-300-40-V-T	501,766	173	501,766	**29**	0.00	470,201	6.29	435,874	13.13
20-300-40-F-T	643,395	122	643,395	**10**	0.00	604,682	6.02	563,965	12.35
20-230-200-V-L	**76,389**	447	76,946	843	0.72	70,046	8.30	59,771	22.32
20-300-200-F-L	**118,756**	865	119,590	755	0.70	105,131	11.47	85,959	28.12
20-300-200-V-T	**76,501**	823	77,858	540	1.74	71,841	6.09	64,765	16.82
20-300-200-F-T	**110,392**	594	110,609	789	0.20	98,684	10.61	83,939	24.11
30-520-100-V-L	54,533	431	**54,454**	186	-0.15	51,598	5.38	48,071	11.72
30-520-100-F-L	**96,311**	813	97,199	851	0.91	87,023	9.64	67,800	30.25
30-520-100-V-T	53,928	110	**53,812**	743	-0.22	46,312	14.12	48,506	9.86
30-520-100-F-T	**99,391**	806	100,871	690	1.47	87,396	12.07	77,265	23.40
30-520-400-V-L	**114,949**	880	117,078	832	1.82	102,044	11.23	101,255	13.51
30-520-400-F-L	**152,863**	859	157,682	899	3.06	131,073	14.25	125,508	20.40
30-520-400-V-T	**116,826**	873	118,214	720	1.17	104,040	10.94	105,347	10.88
30-520-400-F-T	**157,929**	894	161,577	800	2.26	136,023	13.87	127,138	21.31
30-700-100-V-L	47,883	435	47,883	**194**	0.00	46,332	3.24	43,843	8.44
30-700-100-F-L	**60,805**	682	61,254	329	0.73	56,091	7.75	48,091	21.49
30-700-100-V-T	**47,670**	217	47,736	502	0.14	43,977	7.75	42,656	10.64
30-700-100-F-T	**56,804**	440	57,125	449	0.56	51,100	10.04	47,904	16.14
30-700-400-V-L	**98,949**	722	102,216	873	3.20	84,346	14.76	83,221	18.58
30-700-400-F-L	**139,375**	886	144,077	850	3.26	114,326	17.97	104,889	27.20
30-700-400-V-T	**98,260**	800	99,118	516	0.87	84,662	13.84	83,513	15.74
30-700-400-F-T	**134,464**	814	138,787	740	3.11	113,731	15.42	108,944	21.50
Average					1.06		9.46		17.88

CHAPTER IV

IP-BASED SEARCH IN PRACTICE

National less-than-truckload (LTL) carriers run high-volume freight transportation operations, often spending millions of dollars in transportation and handling costs each week. A LTL motor carrier transports shipments that typically occupy only 5-10% of trailer capacity. Hence, transporting each customer shipment directly from origin to destination is not economically viable. As a result, LTL carriers collect and consolidate freight from multiple shippers to increase trailer utilization, referred to as the "load factor," and route freight through a network of consolidation terminals between origin and destination. The savings generated by increasing trailer load factors through consolidation is partially offset by other costs; transferring freight between trailers generates a handling cost, and terminal-to-terminal routing increases the total time and distance each shipment requires to reach its destination.

This chapter focuses on methods for planning how freight should be routed from origin to destination through the terminal network, specifying where along the way it is transferred from one trailer to another. In the LTL industry, this service network design problem is known as constructing a *load plan*. Customer service and pricing pressures have made effective load plans more critical than ever to a carrier's success. National LTL carriers now compete with both so-called super-regional carriers (resulting from mergers of regional LTL carriers), and also LTL service offerings from traditional package express companies like UPS and FedEx; this has led to pressure on pricing, as well as increased competition on customer service including transit time and on-time reliability.

LTL shipments are quoted a service standard from origin to destination in business days. Historically these standards were long enough (often 5 or more business days) that service only loosely constrained freight routing decisions. Today, service standards of 1, 2, and 3 days are much more common (Figure 3 presents a freight profile by service standard for

a national carrier), and these tighter standards must be enforced when planning shipment paths. Shorter service standards reduce opportunities for consolidation (since consolidation introduces handling time and circuity time penalties), especially if rigid operating rules are also enforced (such as a shipment transferred at most one breakbulk per day). Modern freight routes are more complex, and a shipment may visit multiple breakbulks in a single operating day. As a result, carriers need methods for designing load plans that accurately model how short service standards constrain shipment paths and the consolidation opportunities that still exist.

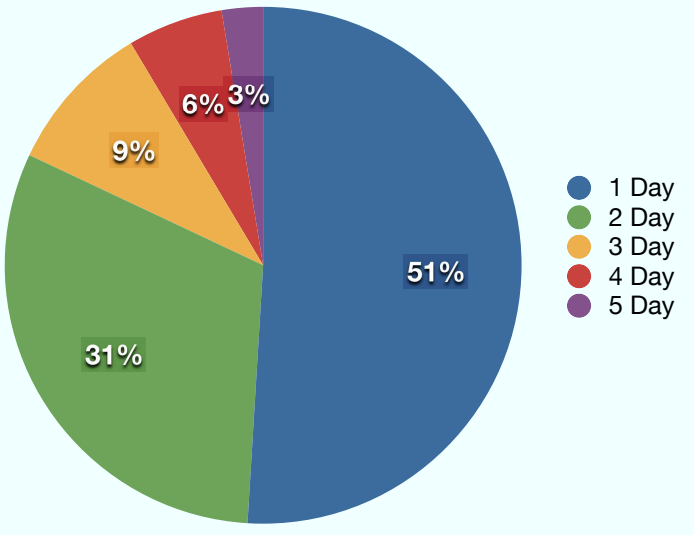

Figure 3: # Shipments by Service Standard

LTL load plans are typically constructed using a number of self-imposed simplifying rules that are later relaxed during actual operations. For example, a traditional load plan assumes that the same freight routing decisions are executed every day. On a given operating day, terminal managers then often override the plan in an attempt to reduce linehaul costs given daily freight volume fluctuations; a simple example is to build a trailer that skips a planned next handling terminal. By relaxing some of these simplifying rules when planning, it may be possible to build improved load plans that generate cost savings and require less manual replanning. For example, one such relaxation would be to plan for predictable daily volume fluctuations by allowing for different freight routing decisions on different weekdays.

In this chapter, we discuss the design and implementation of load planning technology that is more accurate than existing solutions, because it explicitly and accurately models time, and more flexible than existing solutions, because it accommodates relaxations to the simplifying rules of traditional load plans. In our approach, we model freight routing decisions on a time-space network. To accurately capture consolidation opportunities given short service standards, we map a single terminal on a single day to multiple nodes. The resulting network is very large with up to 6,000 nodes and 500,000 arcs. To generate plans that allow different freight routing decisions by weekday, we model freight originating at a terminal on a given day and destined for another terminal on another day as a commodity, resulting in instances with more than 60,000 commodities. To effectively solve such large instances, we develop an approach that combines exact optimization with heuristic search. In particular, we implement a neighborhood search approach in which neighborhoods are defined by the feasible solutions to an integer program, and searching for an improving solution in the neighborhood is conducted by solving the integer program.

Our load planning technology differs in one other important aspect from existing solution approaches. Due to the difficult nature of load plan design, existing approaches simplify the problem by separating the freight routing decisions from empty trailer repositioning decisions (repositioning is necessary due to freight demand imbalance). By sequentially deciding first how to route freight, and then how to move trailer equipment to eliminate imbalances, traditional approaches lead to freight routing decisions that ignore natural "backhaul lanes" where empty trailers are likely to be moving. Carriers correctly understand that it is desirable to route freight when possible along such lanes, and often must modify traditional load planning outputs to do so. Our load planning technology makes freight routing and empty repositioning decisions simultaneously.

The research presented in this chapter makes contributions both in the context of load plan design and heuristic search. Specifically, we present methodology that

- designs traditional load plans that offer significant cost-savings; approximately $200,000 weekly for a national carrier,

- is the first to model time using a discretization appropriate for the tight service standards a carrier must offer to remain competitive,
- is the first to integrate empty trailer repositioning trailers within freight routing decisions,
- designs day-differentiated load plans that account for predictable daily freight volume fluctuations that yield even greater cost-savings; approximately $350,000 weekly for a national carrier, and
- illustrates a successful application of using exact optimization within heuristic search for a large-scale optimization problem.

The remainder of the chapter is organized as follows. Section 4.1 provides background information on the LTL industry and load plan design in particular, and Section 4.2 presents a brief review of relevant literature. Section 4.3 details how our model of LTL operations represents a significant improvement over existing models in the literature. Section 4.4 outlines how we model freight routing, and Section 4.5 presents an integer programming formulation of the load plan design problem and some of its variants. Section 4.6 details the solution heuristic, including integer programming techniques to improve its performance. Finally, Section 4.7 presents the results of an extensive computational study conducted using data from a national LTL carrier.

4.1 Background

LTL networks include two types of terminals: *satellite* or *end-of-line* (EOL) terminals that serve only as origin or destination terminals for freight, and *breakbulk* (BB) terminals that additionally may serve as transfer points for shipments. The set of EOL and BB terminals is known as the *linehaul* network, and typically has a hub-and-spoke structure with an EOL loading to at most two or three BB terminals. A separate operation from each terminal, known often as the *city operation*, is responsible for pickup and delivery at customer locations.

Shipments are consolidated at two levels of the linehaul network. During the day, city operation drivers deliver and collect shipments from customers within a small geographic area. Collected shipments are then brought to the terminal serving as the sorting center and loading facility for outbound and inbound freight for the geographic area. This is the first level of consolidation.

When these origin terminals do not have enough freight to build a full trailer to a destination terminal, they instead route freight to an intermediate BB terminal; this is the second level of consolidation. For example, outbound freight from an EOL is typically loaded first to a BB terminal where it is consolidated with freight from other EOLs and BBs into outbound trailers. The outbound trailers from the BB may be built direct to certain destinations, or may be built to other downstream BBs for another round of consolidation. The consolidation operation at BBs involves unloading and reloading of trailers, and thus involves both time and cost (commonly referred to as *handling* time and cost).

An originating shipment is typically delivered by the city operation to the origin terminal by 7 or 8 pm, and must be moved to the destination terminal by 8 or 9 am on the day of delivery specified by the service standard (all times local). For an example drawn from a national LTL carrier, a shipment originating in Atlanta, GA on Monday with a destination of Cincinnati, OH and a service standard of 1 business day will be available at the Atlanta terminal on Monday by 7 pm and must be moved to the Cincinnati terminal by no later than 8 am Tuesday morning.

An important concept in load planning is that of a *direct*. A direct specifies where handling of freight occurs, and is a fundamental building block for a load plan. If a shipment uses the direct A $\rightarrow$ B, it is loaded into a trailer at A which is not opened (and hence the freight not handled) again until it reaches B. Note that a direct trailer may be moved by multiple drivers (and possibly, multiple modes) en route from A to B. For example, a trailer on the direct Atlanta, GA $\rightarrow$ Seattle, WA may actually travel Atlanta, GA $\rightarrow$ Nashville, TN $\rightarrow$ Kansas City, MO $\rightarrow$ Spokane, WA $\rightarrow$ Seattle, WA; in this example, the trailer is moved by rail from Kansas City to Spokane.

Consider a path from origin to destination consisting of a sequence of directs. Then, a

load plan specifies a path for each shipment and thus prescribes how all freight should be routed through the linehaul network. A typical traditional load plan will specify a unique path for each (origin terminal, destination terminal) combination. Furthermore, such plans also have a special structure where the set of all paths terminating at a specific destination terminal d form a directed in-tree on the network of potential directs. Thus, all shipments that pass through intermediate terminal i on a path to d use the outbound direct (i, j). For example, the load plan may give the following instruction: "all freight in Jackson, TN destined for Atlanta, GA loads direct to Nashville, TN."

Of course, freight cannot move without a trailer to carry it, hence an LTL carrier usually needs to move trailers empty to correct trailer imbalances caused by underlying freight volume directional imbalances. Empty trailer repositioning movements incur costs. Furthermore, load plans are typically adjusted in practice to attempt to route freight when possible along natural backhaul lanes. While such adjustments may reduce the *load factor* of loaded trailers (the fraction of available space in trailers that is occupied by shipments), they can reduce total system costs.

The load plan design problem is to determine how freight is routed through the network of potential directs. Load plans are designed to minimize total linehaul costs which are broken down into two categories:

1. Transportation costs associated with moving loaded and empty trailers

2. Handling costs associated with transferring freight between trailers at a terminal.

A traditional load plan specifies freight paths that do not vary during the week. However, the freight volumes seen by carriers often exhibit predictable daily variation, either in the freight volume or the direction of the freight. For example, a carrier may serve a customer whose distribution center is in Lexington, KY. On Monday, outbound freight for that customer may be destined for terminals in the Southeast, while on Tuesday it may be destined for terminals in the Northwest. Additionally, since service standards to customers are quoted in business days, shipments that originate near the end of the business week often can be routed over many more time-feasible paths because of the flexibility offered by

the weekend.

Hence, we introduce in this chapter the concept of a *day-differentiated load plan*. A day-differentiated plan will specify a unique path for all freight from a specific origin to a specific destination that originated on the same weekday. To do so, we extend the in-tree structure of a traditional load plan such that the outbound direct (i, j) for freight at i destined for d also depends on the day of the week. For example, a day-differentiated load plan could specify: "on Monday all freight in Jackson, TN destined for Atlanta, GA loads direct to Nashville, TN, but on all other days all freight in Jackson, TN destined for Atlanta, GA loads direct to Birmingham, AL."

4.2 Literature Review

Load plan design is similar to the classic network design problem, which has been extensively studied. A common theme in much of the work on network design is that exact optimization is often impractical for realistically-sized instances, and therefore heuristic solution techniques are needed in practice. Metaheuristics have been developed that find good primal solutions to instances of the capacitated fixed charge network design problem. A tabu search algorithm using pivot-like moves in the space of path-flow variables is proposed in [10]. The scheme is then parallelized in [8]. A tabu search algorithm using cycles that allow the re-routing of multiple commodities is presented in [15]. This cycle-based neighborhood is incorporated within a path-relinking algorithm in [16]. Each of these heuristics allows the flow for a specific commodity (defined by an (origin, destination) pair) to be split among multiple paths. Furthermore, none considers underlying equipment moves (including empty repositioning movements).

Load plan design can be seen as a special case of service network design; this problem class has also received a great deal of attention (see [9] or [37] for a review of research in this area). The need to consider equipment management decisions in service network design problems is recognized in [25], which presents both a model and a metaheuristic for the problem. However, the instance sizes considered are significantly smaller than those typical for load planning for a large LTL carrier, and it is not clear how effective the proposed

solution approach would be if adapted to the load plan design problem.

Relatively little research has focused specifically on the load plan design problem for LTL carriers. Early research focused on problem models developed using flat (static) networks that do not explictly capture service standard constraints or the timing of consolidation opportunities. A local improvement heuristic for such a model is presented in [28]; related work includes [30], [31], and [29]. Recognizing the limitations of the static network models, [32] present a dynamic model that can more accurately model consolidation timing. The paper presents an alternative heuristic that relies on determining service network arc subgradients by solving large-scale multi-commodity network flow problems. However, this approach allows origin-destination shipments to split onto multiple paths and does not model empty equipment balancing decisions. Most recently, [21] present a model that more accurately captures consolidation timing using a model that is structurally similar to ours. However, time is modeled much more coarsely with a single node per day per terminal, and loaded and empty routing decisions are considered separately and sequentially. To the best of our knowledge, no work has allowed freight routing decisions to vary by day.

The neighborhood search solution heuristic that we develop in this chapter solves carefully chosen integer programs to improve an existing solution. Thus, like very-large-scale neighborhood search heuristics (VLSN) it uses exponential-sized neighborhoods ([2]). However, in contrast to VLSN, no polynomial-time algorithm exists for searching these neighborhoods. A similar approach is developed for the classic network design problem in Chapter 2.Other examples of combining exact and heuristic search techniques can be found in [13], [36], [3], and [35].

4.3 Enhanced Load Planning

To improve model-based approaches for LTL load planning, we focus on three key modeling features that differ significantly from what is found in existing load plan research: we model time explicitly at an appropriate level of detail, we consider empty and loaded trailer routing decisions simultaneously, and we enable creation of plans that allow additional routing flexibility by relaxing one of the primary assumptions of traditional load plans.

Regarding the third key feature, the primary relaxation we consider is to allow generation of day-differentiated load plans.

- **Modeling Time:** Approximately 80% of the freight volume for the carrier that we studied has a service standard of either 1 or 2 business days. The need to build consolidated shipments in conjunction with such short service standards necessitates a model that uses a time discretization with a finer level of detail than typically considered. According to our carrier's existing load plan, nearly 20% of overnight (one-day service standard) shipments were transferred from one trailer to another at least once. For example, consider the following existing path for freight originating in Evansville, IN at 7 pm destined for Atlanta, GA at 8 am the next day:

  - Travel for 3 hours from Evansville, IN to Nashville, TN.
  - Handling for 2 hours in Nashville, TN.
  - Travel for 5 hours from Nashville, TN to Atlanta, GA.

  In addition, consider the path for freight originating in Louisville, KY at 8 pm also destined overnight to Atlanta:

  - Travel for 2.5 hours from Louisville, KY to Nashville, TN.
  - Handling for 2 hours in Nashville, TN.
  - Travel for 5 hours from Nashville, TN to Atlanta, GA.

  First, we make a simple observation: a time-space model that discretizes time with a single node for each terminal each day and arcs that only move forward in time would conclude that each of these paths is infeasible, and thus a potential real-world consolidation opportunity would be lost. However, by modeling time at a finer level of detail, it is possible to determine dispatch times for each of these paths such that the freight may be consolidated at Nashville into a common trailer (or trailers) outbound to Atlanta.

  Freight is only handled and consolidated at EOL terminals when it initially enters the linehaul system, and is deconsolidated at those terminals only when it exits linehaul

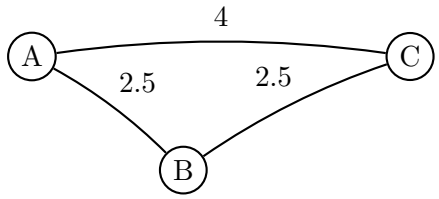

Figure 4: Example Network

for delivery. In our models, we assume that all freight enters the linehaul network (sorted and ready for dispatch) at 7 pm and must arrive at the destination terminal by 8 am. Therefore, we only include nodes for these two time epochs for EOLs in our model. Freight is handled at consolidated at BB terminals primarily during the overnight hours, therefore we divide a day into the windows {1 to 3 am, 3 to 5 am, 5 to 8 am, 8 am to 2 pm, 2 to 7 pm, 7 to 9 pm, 9 to 11 pm, 11 to 1 am}, with nodes specified at each time breakpoint.

- **Empty Integration:** Since solving realistically-sized instances of models for the load plan design problem is beyond the capability of even the most sophisticated integer programming solvers available today, heuristic solution approaches are used. A common feature in all approaches is to first route the freight (hence determining requirements for loaded trailers) assuming trailers are always available when needed. Then, a linear program is solved to resolve trailer imbalances (specifically, a transportation problem).

  Such sequential heuristics, however, may result in freight routing decisions that carriers would never implement. For example, consider the small static network presented in Figure 4 where the numbers above each arc represent the dispatch cost of a trailer. Suppose that we have the following freight, where volume is measured in fractional trailerloads:

  - One originating at C and destined for A,
  - One originating at C and destined for B,
  - One half originating at A and destined for C, and

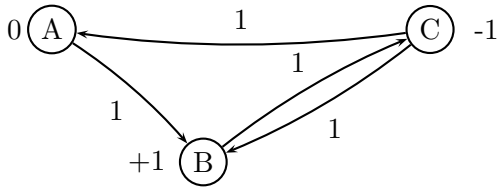

Figure 5: Freight routing decisions yield the following trailer movements when empty costs are not considered.

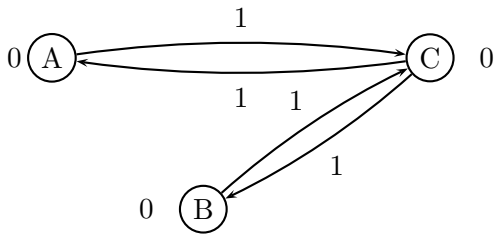

Figure 6: Freight routing decisions made in conjunction with empty repositioning decisions yield higher loaded costs, but lower total costs.

– One half originating at B and destined for C.

Given full trailerloads traveling $C \rightarrow A$ and $C \rightarrow B$, there is no need to consolidate that freight. If we ignore empty repositioning and only minimize the costs of moving loaded trailers to determine freight routing, we obtain the solution in Figure 5, where the numbers above arcs represent the number of trailers dispatched, and the numbers beside each node represent the balance of trailers at that node. The loaded trailer cost is 11.5, but resolving trailer imbalances at B and C costs 2.5 for a total linehaul cost of 14. If on the other hand, we simultaneously optimize freight routing and trailer repositioning, we obtain the solution in Figure 6, which has a higher loaded trailer cost (13), but requires no repositioning of empty trailers and hence has a lower total linehaul cost. In practice, a carrier would recognize that $A \rightarrow C$ and $B \rightarrow C$ are backhaul lanes and hence would be wary of solutions like those in Figure 5 that move freight away from those lanes.

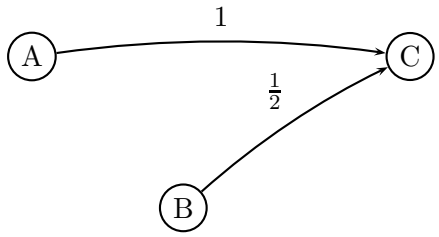

Figure 7: Load plan on Monday does not consolidate freight.

- **Day-Differentiation:** The first two proposed improvements address the accuracy of how traditional LTL carrier operations are modeled. Alternatively, enabling day differentiation of load plans allows us to evaluate the potential cost savings of relaxing the rules that govern how a carrier routes freight. Consider again our example network in Figure 4, but now assume the following freight volumes:
  - **On Monday**:
    - One originating at A and destined for C,
    - One half originating at B and destined for C.
  - **On Tuesday, Wednesday, Thursday, Friday**:
    - One originating at A and destined for C,
    - One originating at B and destined for C.

  Given a full trailer traveling $A \rightarrow C$ on Monday, there is no need for consolidation, and the load plan depicted in Figure 7 (the numbers above each arc are the fractional trailerloads assigned to each arc) is optimal. For the other days of the week, however, it is less costly to route the $A \rightarrow C$ freight through B and consolidate. Thus, it is better to execute the load plan depicted in Figure 8 on Tuesday through Friday. In fact, most carriers already route freight in a day-differentiated manner, deviating from the prescribed load plan when opportunities arise. On Monday, the terminal manager at A will recognize that there is enough freight going $A \rightarrow C$ to build a full trailer and send it on the *opportunistic direct* $A \rightarrow C$ even though the load plan

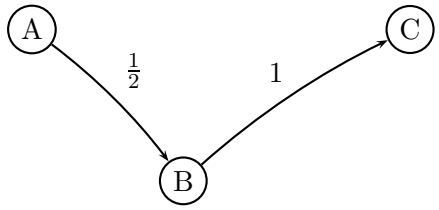

Figure 8: Load plan on Tuesday-Friday consolidates freight through BB terminal b.

may specify that the freight should next be routed to B. By enabling the creation of day-differentiated plans, we create the ability to capture some of the daily freight routing decisions made by terminal managers, and potentially find improvements. In practice, terminal managers typically base their decisions about when to introduce an opportunistic direct solely on information available locally at their terminal. Thus, the decision to use an opportunistic direct could actually lead to increased downstream costs. By incorporating day-differentiation into our model, we can capture the global impact of these decisions.

4.4 Modeling Freight Routing

To design a load plan, we must determine the directs used to route freight as well as when and where to hold freight to improve consolidation and reduce system costs. To make these decisions, we model terminals and potential directs on a time-space network. Let $LN = (U, L)$ denote the carrier's linehaul network, where U is the set of terminals in the carrier's network and L is the set of potential directs connecting terminals. Associated with each direct $l = (u_1, u_2) \in L$ is a transit time that reflects how long it takes a carrier to route a trailer from terminal u_1 to terminal u_2. For a given time discretization of a planning horizon, we define the time-space linehaul network $TS - LN = (N, A)$, where N denotes the set of nodes and A denotes the set of arcs. Each node $n = (u, t), u \in U, t \in T$ represents a terminal at a particular point in time. Each arc $a = ((u_1, t_1), (u_2, t_2))$ with u_1 and $u_2 \in U$ and $u_1 \neq u_2$ represents a potential dispatch from u_1 at time t_1 on direct (u_1, u_2) arriving in u_2 no later than time t_2. We create such arcs for each direct $l = (u_1, u_2) \in L$ and each

timed copy (u_1, t_1) of the origin node u_1. The destination node (u_2, t_2) is then chosen to be the earliest timed copy of the node u_2 such that $t_2 - t_1$ is no less than the transit time of the underlying direct l. We also create arcs $a = ((u_1, t_1), (u_1, t_2))$ to connect subsequent timed copies of each node u_1. These allow us to model holding a trailer or shipment at terminal u_1.

Given networks LN and $TS - LN$, let $\delta^+(u) \subseteq L$ denote the set of potential outbound directs from terminal $u \in U$, $l(a)$ denote the direct $l \in L$ corresponding to the arc $a \in A$, $\delta^+(n) \subseteq A$ denote the set of outbound arcs from node $n \in N$, $\delta^-(n) \subseteq A$ denote the set of inbound arcs to node $n \in N$, and c_a denote the per-trailer travel cost along arc $a \in A$.

We model freight which enters the linehaul network at terminal u_1 on day d_1 as entering the time-space network at node $n_1 = (u_1, t_1)$ where $t_1 = d_1$ @ 7 pm. Freight which must reach terminal u_2 by day d_2 is given the destination node $n_2 = (u_2, t_2)$ where $t_2 = d_2$ @ 8 am. All freight shipments with a common (n_1, n_2) pair are considered a single *commodity*. Carriers typically quote a single service standard for freight originating at terminal u_1 and destined for terminal u_2. Although we use this assumption, accommodating multiple classes of service (say "regular" and "express") between u_1 and u_2 is also enabled by this modeling framework.

Let K denote the set of commodities. For each commodity $k \in K$, let $o(k)$ denote the origin terminal, $d(k)$ denote the destination terminal, $on(k) \in N$ denote the origin node, $dn(k) \in N$ denote the destination node, w_k denote the weight in pounds, and q_k denote its size measured in fractional trailers (note, q_k need not be less than one). Let $K(d) \subseteq K, d \in U$ denote the set of commodities with destination terminal d and let $K(o, d) \subseteq K, o \in U, d \in U$ denote the set of commodities with origin terminal o and destination terminal d.

Our approach will use a path-based optimization model on the time-space network. Therefore, let $P(k)$ be a set of possible freight paths for commodity $k \in K$, where a freight path p is a sequence of arcs, i.e., $p = (a_1, \ldots, a_{n_p})$. Each path $p = (a_1, \ldots, a_{n_p}) \in P(k)$ is constructed such that $a_1 \in \delta^+(on(k))$ and $a_{n_p} \in \delta^-(dn(k))$. How commodity k is routed then simply becomes a question of choosing a path $p \in P(k)$. By using a path-based model, certain constraints can easily be enforced, *e.g.*, freight is handled at most

two times. Associated with a path $p = (a_1, \ldots, a_{n_p})$ is an underlying path p of directs $\mathsf{p} = (l(a_1), \ldots, l(a_{n_p}))$. For convenience, we sometimes represent a path p of directs as a sequence of terminals, *e.g.*, we may refer to the path of directs $((u_1, u_2), (u_2, u_3))$ as (u_1, u_2, u_3). Note that given a path p, we can calculate its total handling cost h_p per pound by summing the costs for the intermediate terminals visited.

To construct a set of paths $P(k)$ for commodity k, we begin by computing up to m minimum cost paths in the linehaul network LN with respect to total travel and handling cost (for some given value of m). Since we ignore time in this computation, we next determine which of these paths on LN can be mapped to service feasible paths on $TS-LN$. To do so, we first determine the minimum execution duration of each path by mapping the sequence of directs to a feasible set of arcs in A, determined by the transit time of the directs and required handling time between directs at intermediate terminals. For commodities that represent 1-day (overnight) services, 30 minutes of handling time is assumed while for all other service standards, we assume two hours.

To insert handling time into a path, we assume that the slack created by mapping transit times to our time discretization is available for handling. Consider a path of directs (u_1, u_2, u_3) for a commodity with 2 day service originating on Monday at u_1 and destined for u_3. Since initial dispatches occur at 7 pm, if the transit time from u_1 to u_2 is 11 hours then the arc in A arrives at u_2 on Tuesday morning at 6 am. Since the next timed copy of u_2 in N occurs at 8 am, we assume handling occurs between 6 am and 8 am, and this commodity is ready for dispatch to u_3 at 8 am on arc $a = ((u_2, Tuesday\ @\ 8am), (u_3, t_3))$. If, however, the transit time from u_1 to u_2 is 12 hours, then only one hour of handling time is available prior to 8 am. Therefore, we next determine whether there is slack available in the mapping of (u_2, u_3) to $TS-LN$. For exposition, suppose that u_3 is a BB terminal, and therefore its next timed copy is Tuesday at 2 pm. Then, if the transit time from u_2 to u_3 is ≤ 5 hours, we assume that the remaining hour of handling occurs before the dispatch and map the direct (u_2, u_3) to the arc $a = ((u_2, Tuesday\ @\ 8am), (u_3, t_3))$.

We recognize that given this mapping methodology, for different paths for different commodities, the arc $((u_2, Tuesday\ @\ 8am), (u_3, t_3))$ may assume a dispatch at slightly

different points in time. Since we will assume that any freight traveling on the same arc $a \in A$ can be loaded into the same trailers, these assumptions might seem to overestimate consolidation opportunities. However, it should be noted that the handling time estimates are not hard lower bounds, and carriers can prioritize handling to reduce these times when needed.

For each path of directs that is service feasible, we include not only the minimum duration path p into $P(k)$, but potentially also other versions that add holding arcs of the form $((u_1, t_1), (u_1, t_2)$ if they are also feasible. Adding such timed copies models the ability to hold freight at intermediate terminals to improve the plan. We construct a limited set of such paths by only holding freight until specific events occur. First, we allow freight to be held at a terminal until the time that new freight originates at that terminal; thus, freight arriving at a breakbulk during the day can be consolidated with that evening's originating outbound freight. Second, we allow freight to be held at a terminal until its cut time, *i.e.*, the latest time at which the freight can be dispatched and still arrive on time to its destination. In this way, freight destined for common destinations may be consolidated.

Consider the two examples in Figure 9. In the first, commodity k originates at and is destined for an end-of-line terminal. The network in the figure depicts the locations and time points where we model consolidation opportunities for this commodity, and all possible paths between the origin node and destination node of the commodity in the network would be added to $P(k)$. In the second example, commodity k originates at and is destined for a breakbulk terminal, and thus there are many more opportunities for this commodity to be consolidated with other freight.

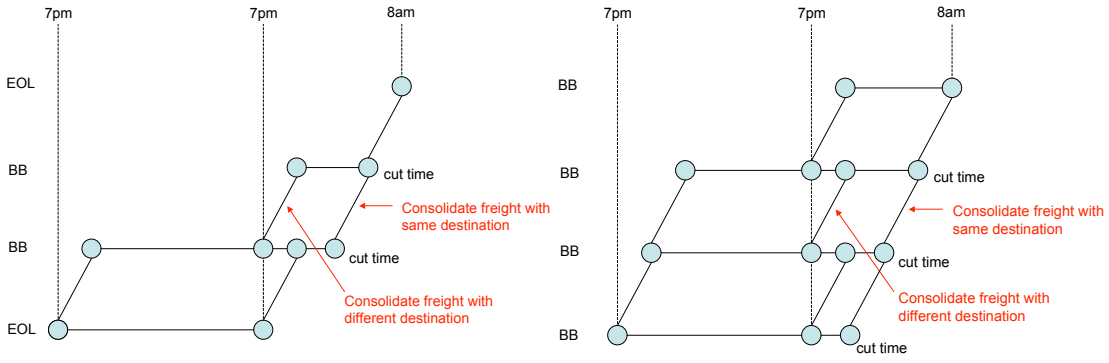

Figure 9: Holding Freight for Consolidation

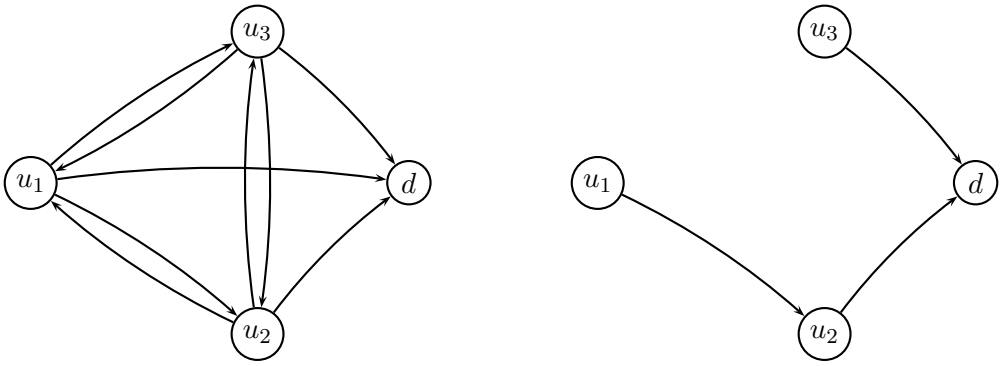

Figure 10: LN for a four terminal network, and load plan as a directed in-tree into d

We have introduced two networks, LN and $TS-LN$, each of which plays a role in load plan design. Recall that a traditional load plan specifies the unique direct a shipment should take given its current *terminal* location and its ultimate *terminal* destination. Choosing the unique outbound direct for freight at terminal u_1 and destined for terminal d (regardless of its origin or service standard) corresponds to choosing a single arc in LN from $\delta^+(u_1)$ for freight destined to node $d \in U$. Hence, the structure of a traditional load plan requires that the directs chosen for freight destined for terminal d must form a directed in-tree on LN rooted at the node d (as depicted on a small example in Figure 10). Note this tree structure also implies that freight at terminal u_1 and destined for terminal d follows a unique path in LN.

In our path-based approach, we choose for each commodity $k \in K$ on $TS-LN$ a path of arcs, where a component arc a is a timed copy of a direct. Therefore, when constructing load plans, we must ensure that the set of paths chosen for all commodities are such that there is appropriate consistency of the paths selected for commodities in $K(d)$ for each given destination d. For traditional load plans, we have chosen only to ensure that the outbound direct $l \in L$ is the same for all such commodities, but the timing of the movements may change. Continuing the example from the previous paragraph, Figure 11 illustrates a simplified time-space network where each direct requires one time period for transit and handling. Furthermore, the depicted network only includes directs consistent with the load plan for commodities k with $dn(k) = (d, t_i)$ given in Figure 10. Suppose freight is destined

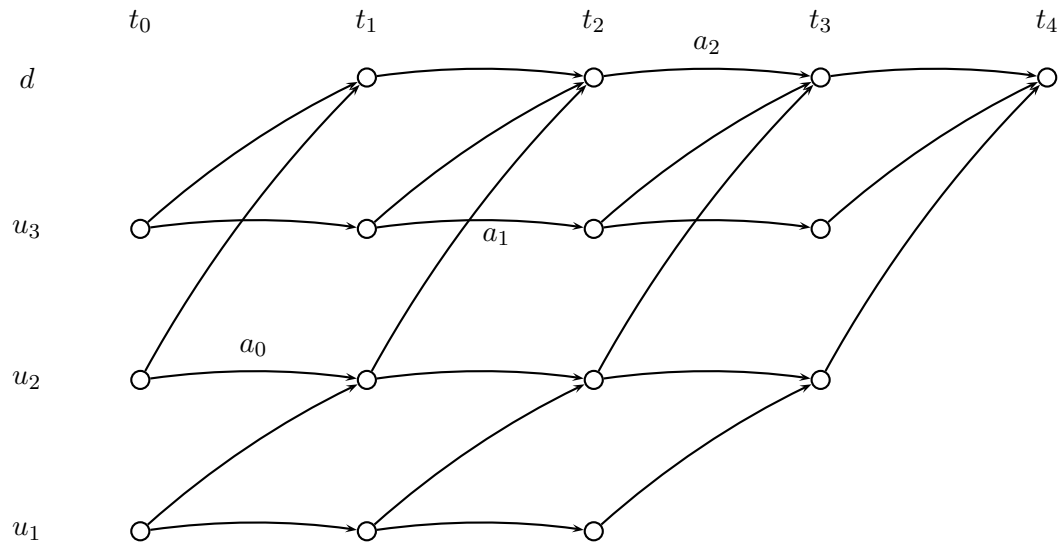

Figure 11: Time-space network depicting routing choices into d given a load plan.

for d originating at u_2 at both t_0 and t_1, with a service standard of two periods. Thus, in our model there are two commodities, $k_0 = ((u_2, t_0), (d, t_2)$ and $k_1 = ((u_2, t_1), (d, t_3))$. For k_0, the model might choose the path $p_0 = (a_0, a_1)$, which holds the freight at the origin for one period, and the path $p_1 = (a_1, a_2)$ for k_1 which does not hold the freight. Such decisions consolidate the freight on arc a_1.

4.5 Load Plan Design Integer Program

We next present an integer programming formulation of the traditional load plan design problem (TLD-IP). TLD-IP has three sets of decision variables. First, x variables indicate whether commodity k uses path p, *i.e.*, $x_p^k \in \{0, 1\} \ \ \forall k \in K, \ \ \forall p \in P(k)$. Second, y variables enforce consistency between paths for commodities heading to common destinations by indicating whether direct $l \in \delta^+(u)$ is chosen for all commodities destined for terminal d routed through terminal u, *i.e.*, $y_l^d \in \{0, 1\} \ \ \forall d \in U, \ \ \forall l \in \delta^+(u), u \in U$. Finally, τ variables count the number of trailers (empty or loaded) that move on arc a, i.e., $\tau_a \in \mathbb{Z}_+ \ \ \forall a \in A$.

The formulation is to then minimize

$$\sum_{a \in A} c_a \tau_a + \sum_{k \in K} \sum_{p \in P(k)} h_p w_k x_p^k$$

subject to

$$\sum_{p \in P(k)} x_p^k = 1 \ \ \forall k \in K \tag{24}$$

$$\sum_{l \in \delta^+(u)} y_l^d \leq 1 \ \ \forall u \in U, \ \ \forall d \in U \tag{25}$$

$$\sum_{p \in P(k): a \in p} x_p^k \leq y_{l(a)}^{d(k)} \ \ \forall k \in K, \forall a \in A \tag{26}$$

$$\sum_{k \in K} \sum_{p \in P(k): a \in p} q_k x_p^k \leq \tau_a \ \ \forall a \in A \tag{27}$$

$$\sum_{a \in \delta^+(v)} \tau_a - \sum_{a \in \delta^-(v)} \tau_a = 0 \ \ \forall v \in V. \tag{28}$$

The objective is to minimize total transportation and handling costs. Constraints (24) ensure that a path is chosen for each commodity. Constraints (25) ensure that a single outbound direct is selected for each terminal u and destined for terminal d. Constraints (26) ensure that a path can only be chosen for commodity k when all of its component directs are chosen. Constraints (27) ensure that there are enough trailers moved along an arc to carry the freight assigned to the arc via the paths chosen. Finally, constraints (28) ensure flow balance of trailers at every node in the time-space network, and thus ensure proper repositioning of trailers.

For reasonably sized instances, there will be prohibitively many constraints (26). However, a valid formulation with fewer constraints, but a weaker linear programming relaxation, can be obtained. Let $\mathsf{P}(k)$ denote a set of paths of directs for commodity $k \in K$, and for a destination $d \in U$, let $K_l(d) \subseteq K$ be the set of commodities $k \in K(d)$ such that $\exists\ \mathsf{p} \in \mathsf{P}(k)$ containing the direct l. Then, by aggregating over the set $K_{l(a)}(d(k))$, the following constraints are equivalent to (26):

$$\sum_{k \in K_{l(a)}(d(k))} \sum_{p \in P(k): a \in p} x_p^k \leq |K_{l(a)}(d(k))| y_{l(a)}^{d(k)} \ \ \forall a \in A.$$

4.5.1 Variations on Traditional Load Plan Design

Without requiring the directs in a load plan to form an in-tree, the problem faced by carriers is a special case of network design. In fact, constraints (24) and (27) represent a variable upper-bound network design problem where the upper bounds represent trailer

capacities. The in-tree requirement is enforced to simplify terminal operations, since it allows a terminal worker to only examine the destination of a shipment to determine the appropriate outbound trailer for loading. However, advances in information technology and the introduction of handheld scanners into terminal operations largely render the in-tree requirement unnecessary for LTL operations.

To understand the cost savings possible by changing the traditional load plan structure, we consider three relaxations: the day-differentiated load plan, the same-path load plan, and the unrestricted load plan.

- **Day-Differentiated Load Plan Design:** Whereas a traditional load plan ensures that a single outbound direct is chosen for freight at terminal u destined for terminal d for the entire week, a day-differentiated load plan only ensures a single outbound direct each day. Day-differentiation can be accommodated in TLD-IP by redefining the variables that indicate what directs are chosen and by slightly modifying the constraints (25) and (26). We will refer to the formulation for the day-differentiated load plan design problem as DDLD-IP. Let the set of days in the planning horizon be denoted by $DAYS$, and let $m(a)$ denote the day $m \in DAYS$ corresponding to the tail node of arc $a \in A$.

  We introduce variables $y_l^{m,d}$ to indicate whether direct $l \in \delta^+(u)$ is chosen for freight destined for terminal d at terminal u on day m, *i.e.*,

  $$y_l^{m,d} \in \{0,1\} \ \ \forall d \in U, \ \ \forall m \in DAYS, \ \ \forall l \in \delta^+(u), u \in U.$$

  The day-differentiated analog of the TLD-IP constraints (25) ensures that a single outbound direct is chosen for freight ultimately destined for terminal d at terminal u on day m, *i.e.*,

  $$\sum_{l \in \delta^+(u)} y_l^{m,d} \leq 1 \ \ \forall u \in U, \ \ \forall m \in DAYS, \ \ \forall d \in U \tag{29}$$

  The day-differentiated analog of the TLD-IP constraints (26) ensures that a path can only be chosen for commodity k when all the underlying day-indexed directs are

chosen, *i.e.*,

$$\sum_{p \in P(k): a \in p} x_p^k \leq y_{l(a)}^{m(a),d(k)} \quad \forall k \in K, \forall a \in A \tag{30}$$

The same comments regarding aggregating the coupling constraints (26) of TLD-IP applies to the constraints (30).

- **Same-Path Load Plan Design:** The same-path load plan drops the in-tree requirement completely but keeps the restriction that the freight between two terminals follows the same sequence of directs every day. We can remove the in-tree restriction from the TLD-IP by removing the variables y_l^d and the constraints (25) and (26).

  Given that we require a unique path between terminals in the linehaul network LN, all commodities with the same origin and destination must use the same sequence of directs. Let $\mathsf{P}(o, d)$ denote the set of paths of directs between terminals $o, d \in U$. Furthermore, let a set of paths of arcs for a path of directs p and commodity k be denoted by $P(\mathsf{p}, k)$. Then, the the following *same-path inequalities* need to be satisfied:

$$\sum_{p \in P(\mathsf{p}, k_1)} x_p^{k_1} = \sum_{p \in P(\mathsf{p}, k_2)} x_p^{k_2} \quad \forall o, d \in U, \ \ \forall k_1, k_2 \in K(o, d), \ \ \forall \mathsf{p} \in \mathsf{P}(o, d) \tag{31}$$

  We will refer to the formulation for the same path load plan design problem as SPLD-IP.

- **Unrestricted Load Plan Design:** The unrestricted load plan drops both the in-tree and same path requirements. We can accommodate this relaxation in the TLD-IP by removing the variables y_l^d and the constraints (25) and (26), leaving a variable upper bound flow path network design problem with the trailer balance constraints 28). We will refer to the formulation for the unrestricted load plan design problem as ULD-IP.

Finally, we note that in practice it may not be practical for a carrier to implement one of these relaxations for its entire terminal network, since the costs of automating all terminal operations with handheld scanners may not be justified by corresponding benefits. It should be clear, however, that it is possible to formulate blended models where only some terminals relax the constraints that enforce single outbound directs for each destination d.

4.6 In-tree Reoptimization Heuristic

Realistically-sized instances of TLD-IP (and its variants) cannot be solved directly by commercial integer programming solvers. Therefore, we develop a local search procedure with neighborhoods defined by carefully chosen restricted versions of TLD-IP; see Algorithm 13 for a general outline of the procedure.

Algorithm 13 Integer Programming Based Neighborhood Search

Require: a feasible solution to TLD-IP
  while the search time has not exceeded a prespecified limit T **do**
    Choose a subset of variables V
    Solve TLD-IP with all variables not in V fixed at their current value
    if an improved solution is found **then**
      Update the best known feasible solution
    end if
  end while

We use a subset of variables V that is motivated by the structural property that the directs selected into a destination terminal d must form a directed in-tree, and we refer to the associated integer program as an In-tree IP into d, or IIP_d. The purpose of IIP_d is to improve the current solution by optimally choosing the directs used for d-bound freight, and by optimally choosing when and where d-bound freight is held. The IIP_d problem is to determine a set of paths for all commodities in $K(d)$; note that this problem is then to determine a new directed in-tree to d. More formally, given a current feasible solution $(\bar{y}, \bar{x}, \bar{\tau})$, IIP_d is defined by holding fixed the variables

- $y_l^u = \bar{y}_l^u \quad \forall u \in U$ such that $u \neq d$
- $x_p^k = \bar{x}_p^k, \forall k \in K \setminus K(d)$.

A specialized version of Algorithm 13 is presented in Algorithm 14. Note that Algorithm 14 can be applied to the traditional or any of its variants by simply modifying the choice of In-tree IP (TLD-IP, DDLD-IP, SPLD-IP, ULD-IP) at each iteration. Note that we never fix the trailer variables τ_a to a certain value. Thus, at each iteration of Algorithm 14 empty trailer repositioning decisions are explicitly considered. Since our approach improves the load plan by re-routing freight destined for a specific terminal, we do not want to spend

Algorithm 14 *IIP* Neighborhood Search

Require: an initial load plan $(\bar{y}, \bar{x}, \bar{\tau})$
  for each terminal d **do**
    Set $F_d = \sum_{k \in K(d)} q_k$, the total amount of freight destined for d
  end for
  Set $TERMS$ = array of top 25% of terminals with respect to F_d
  Set $N = |TERMS|$
  Sort $TERMS$ in descending order of F_d
  Set $iter = 0$
  while the search time has not exceeded a prespecified limit T **do**
    Choose destination terminal $d = TERMS[iter \ \ mod \ \ N]$
    Solve In-tree IP IIP_d
    if Solution to IIP_d gives lower total load plan cost **then**
      Update $(\bar{y}, \bar{x}, \bar{\tau})$
    end if
    Set $iter = iter + 1$
  end while

time solving In-tree IPs for terminals for which little freight is destined. Thus, we only consider the top 25% of terminals for which freight is destined. The algorithm we present iterates through this subset of terminals in a round-robin manner.

The success of Algorithm 14 depends on the time needed at each iteration to solve In-tree IPs. To reduce these solution times, we use two sets of valid inequalities derived from the structure of a load plan and the fact that some variables are fixed when solving an In-tree IP. We also utilize a preprocessing rule based on the knowledge we have of where loaded trailers may flow to prune arcs from $TS - LN$.

Path-continuation inequalities. Path-continuation inequalities are derived from the in-tree structure of a load plan. Suppose that freight originating in Athens, GA and destined for Columbus, OH uses the path of directs $(Athens, Atlanta, Cincinnati, Columbus)$. Then freight originating in Atlanta, GA and destined for Columbus, OH must use the path of directs $(Atlanta, Cincinnati, Columbus)$. Let the first and second in a path p of directs be denoted by $f(\mathsf{p})$ and $s(\mathsf{p})$ respectively. Then, we have the following valid path-continuation inequalities

$$\sum_{k' \in K(o(k),d(k))} \sum_{p \in P(\mathsf{p},k')} x_p^{k'} \leq \frac{|K(o(k),d(k))|}{|K(s(\mathsf{p}),d(k))|} \sum_{k' \in K(s(\mathsf{p}),d(k))} \sum_{p \in P(\mathsf{p} \setminus f(\mathsf{p},k'))} x_p^{k'} \quad \forall k \in K, \ \forall \mathsf{p} \in \mathsf{P}(k). \tag{32}$$

Note that while we have presented the path-continuation inequalities in the context of solving an In-tree IP, they are also valid for the TLD-IP. They can also easily be extended to DDLD-IP.

Trailer Disaggregate Inequalities. Trailer disaggregate inequalities are derived from the fact that the paths for some commodities are fixed when solving an In-tree IP. Let f_a denote the amount of freight that is fixed on arc a due to fixed commodity paths. For IIP_d, we have $f_a = \sum_{k \in K \setminus K(d)} \sum_{p \in P(k): a \in p} q_k \bar{x}_p^k$ and constraints (27) become

$$\sum_{k \in K(d)} \sum_{p \in P(k): a \in p} q_k x_p^k + f_a \leq \tau_a \ \ \forall a \in A. \tag{33}$$

We first observe that when solving IIP_d, we can bound the variable τ_a from below by recognizing that we must have enough trailers to carry the freight that is fixed on the arc a, i.e., $\tau_a \geq \lceil f_a \rceil$. By also considering the freight that we are trying to route over the arc, we can strengthen the inequality. This gives the following trailer disaggregate inequalities

$$\tau_a \geq \lceil f_a \rceil + (\lceil f_a + q_k \rceil - \lceil f_a \rceil) \sum_{p \in P(k): a \in p} x_p^k \quad \forall a \in A, \ \ \forall k \in K. \tag{34}$$

Since we potentially have a trailer disaggregate inequality for each arc and each commodity, it is clearly impractical to add all of them to IIP_d. Hence, we need to determine which might be most effective. Note the derivation of the trailer disaggregate inequality is conducted by replacing a sum over all commodities with a single commodity. Therefore, the inequality will be the strongest on arcs carrying only freight for the chosen commodity k. When we consider any outbound arc from a breakbulk terminal, it is likely that the arc carries freight associated with many commodities. On the other hand, when we consider an outbound arc from an end-of-line terminal, it is likely that the arc carries freight associated with relatively few commodities. Since it is more likely that trailer disaggregate inequalities will be effective on such outbound arcs, we add them to IIP_d only for outbound arcs from end-of-line terminals.

Preprocessing. Since the size of the time-space network can get large quickly for practical instances, reducing the network size, *i.e.*, eliminating arcs (or flows on arcs), may significantly enhance our ability to solve instances. Certain arcs in the network can only be

used for empty repositioning, and often there exist alternate times at which this repositioning can take place. Recognizing this, we can restrict the number of repositioning options. This not only reduces the size of the instances that need to be solved, but also eliminates some of the symmetry embedded in the instances. More specifically, if there is an arc $a = ((u_1, t_1), (u_2, t_2)) \in A$ where node (u_2, t_2) does not appear in any path for any commodity, then there is no reason to use a for repositioning as there always exists an alternate repositioning option with the same cost.

4.7 Computational Results

The algorithm was developed in C++ with CPLEX 11 as the Mixed Integer Program (MIP) solver with which we interfaced via ILOG Concert Technology. When solving MIPs with CPLEX, we set the MIPEmphasis parameter to integer feasibility and all other parameters to their defaults. When solving instances of the In-tree IP, we use an optimality tolerance of .1%; this typically represents approximately $5,000 for the carrier we study. All experiments were run on a Debian Linux computer with 32 GB of RAM and 8 Intel Xeon 2.6 GHz processors. We report all times in seconds.

Using a planning horizon of a week for load planning is very typical. Our test set consists of seven instances based on historical data, each consisting of the freight originating for a single week. The instances represent actual freight volumes transported by a super-regional LTL carrier in the U.S., and include three weeks in April of 2008 (Apr08-W1, Apr08-W2, Apr08-W3) and four weeks in March of 2009 (Mar09-W1, Mar09-W2, Mar09-W3, Mar09-W4). When modeling freight which originates in one week, but is due in the subsequent week, we use a "wrapped" version of the time-space network where arcs connect later time periods in the week to time periods in the beginning.

4.7.1 Solving In-tree IPs

We first focus on configuring and tuning the process of solving In-tree IPs. Specifically, we investigate the benefits derived from the three classes of valid inequalities, i.e., trailer disaggregate inequalities (TD), path-continuation inequalities (PC), and same-path inequalities (SP). Note that the same-path inequalities may improve solution times since they are valid,

even though any load plan satisfying the in-tree structure requirement automatically satisfies them. We measure the effectiveness of classes of valid inequalities by the decrease in optimality gap at the root node of the search tree, the number of integer programs that can be solved in a fixed amount of time, and the savings obtained in a fixed amount of time. For this analysis, we use instance Apr08-W1.

Recall that our neighborhood search approach solves In-tree IPs for the top 25% of destinations in terms of inbound freight, in this case 40 terminals. In the results presented below, the values are averaged over the instances defined by these 40 terminals. Since we have the load plan that was used by the carrier for the test week, we have an initial solution for each IIP_d and hence an upper bound ub_d on the optimal value of IIP_d. The root relaxation value reported by CPLEX provides a lower bound lb_d on the optimal value of IIP_d and thus an optimality gap $gap^{init} = (ub_d - lb_d)/ub_d$. When we add classes of valid inequalities to IIP_d, we obtain an improved lower bound at the root and thus an improved optimality gap. In Table 11, we report the number of rows, the improvement in optimality gap, and the time to solve the root relaxation when using certain classes of valid inequalities.

Table 11: Optimality Gap Reductions

	# Rows	Gap Reduction	Root Solve Time
No Inequalities	37,252	-	21.35
TD	37,438	1.02	21.67
TD + PC	39,400	4.71	23.50
TD + PC + SP	44,587	12.88	27.30

We see that as we add classes of valid inequalities, the optimality gap gets smaller, but with an increase in the time required to solve the root relaxation that is likely due to the increased number of rows.

While smaller optimality gaps are desirable, we are primarily interested in reducing the time required to solve IIP_d instances, since this will allow us to solve more of them in a fixed amount of time. Thus, we next report in Table 12 how many IIP_d instances can be solved to within .1% of optimality in three minutes. The "Number Solved" column reports how many of the 40 IIP_d instances could be solved. The "Avg. Time" column reports the average time it took to solve the instances that were solved. Finally, the "Avg. Time*" column reports the average time it took to solve the 27 instances that could be solved

without adding any of the valid inequalities.

Table 12: Number IIP_d Solved to within .1%

	Number Solved	Avg. Time	Avg. Time*
No Inequalities	27	46.41	46.41
TD	27	40.74	40.74
TD + PC	29	38.97	35.44
TD + PC + SP	30	48.10	40.89

We see that for each configuration of classes of valid inequalities, the average time required to solve the 27 instances that could be solved without adding any valid inequalities reduces. Furthermore, we see that adding path-continuation plus trailer disaggregate inequalities increases the number of instances that can be solved and reduces the average time required to solve the 27 instances that could be solved without adding any valid inequalities by nearly 25%. Although additionally including the same-path inequalities does enable the solution of one more instance, it increases the average time required to solve the 27 instances. This is likely due to the increase in solve time of the root relaxation.

The ultimate goal of adding classes of valid inequalities is to solve more IIP_d instances in a fixed amount of time, which hopefully leads to greater savings. Hence, we next report in Table 13 the savings that are obtained when running the neighborhood search for thirty minutes, where savings are defined as the (cost of the initial load plan minus the cost of the load plan designed by the algorithm) divided by the cost of the initial load plan.

Table 13: % Savings in thirty minutes

No Inequalities	TD	TD + PC	TD + PC + SP
1.54	1.81	2.01	1.82

We see that again the Path-continuation and Trailer Disaggregate inequalities yield the best performance; enabling nearly .5% more savings to be found than when no cuts were added.

Based on the results of these experiments, we chose to explicitly add the trailer disaggregate and path-continuation inequalities to the IIP_d formulation. Alternatively, the same-path inequalities were included as CPLEX *user cuts* in all further experiments.

The integer program IIP_d represents a special type of network design problem, a class of optimization problems which is notoriously difficult to solve due to weak dual bounds. Yet,

we are able to solve instances with more than 20,000 constraints and more than 300,000 variables (after CPLEX preprocessing) in less than one minute. Much of this success can be attributed to the freight that is fixed on arcs and effective preprocessing. In fact, even without adding any valid inequalities the average optimality gap is only .52% for the instances used to produce the results reported in Table 11. However, the root relaxation value used to compute the optimality gap is significantly higher than the value of the linear programming relaxation (LPR) of the original formulation. For the instances used to produce the results reported in Table 11, the average optimality gap when computed using the the optimal value of LPR is 54.74%. Network design problems often exhibit large optimality gaps due to the fact that a solution to LPR only pays for exactly the capacity that is used. By recognizing (in preprocessing) that the number of trailers required on arc a to carry the fixed freight f_a is $\lceil f_a \rceil$, the bound produced by LPR can be strengthened significantly. Adding these bounds reduces the average gap from 54.74% to .65%. Hence, we see that even though the approach relies on repeatedly solving instances of the fixed charge capacitated network design problem, we avoid much of the difficulty associated with solving this type of integer program because much of the fixed charge is already paid.

4.7.2 Traditional Load Plan Improvements

The primary goal of our research is to develop technology that can produce more cost-effective load plans. In Table 14, we report the savings ("$\Delta Cost$") and the increase in pounds per trailer ("ΔPPT") when we run the neighborhood search on the April 2008 and March 2009 instances for six hours. Again, these improvements are measured in percentages relative to the initial load plan provided by the carrier for the relevant week. For these experiments, we provided CPLEX a time limit of 90 seconds to solve each IIP_d instance. Since a 1% savings represents about $60,000 per week for the carrier, these suggested changes indicate a substantial improvement to the carrier's bottom line. The increase in pounds per trailer indicates that the neighborhood search does increase consolidation and thus finds significant savings for each of the weeks. While the approach finds savings in both data sets, they are greater in the April 2008 weeks. We believe that this can be attributed to the

carrier's load plan yielding a higher pounds per trailer in March 2009 than in April 2008.

Table 14: Load Plan Savings For Each Week

	Apr08-W1	Apr08-W2	Apr08-W3	Mar09-W1	Mar09-W2	Mar09-W3	Mar09-W4
$\Delta Cost$	4.49	3.94	4.77	3.73	3.29	3.69	3.77
ΔPPT	2.17	1.37	3.13	2.37	1.68	1.66	1.27

One of the important features of our approach is that it integrates the planning of loaded and empty trailer movements. To analyze the value of such joint planning, we compare the load plans resulting from our approach with those constructed via a sequential approach, *i.e.*, an approach in which freight is routed first (hence determining flows of loaded trailers) and trailer balance is restored second via repositioning. To implement such a sequential approach, we first run our neighborhood search with optimization problems that do not include constraints (28). The result is a solution (x^1, y^1, τ^1) which is not necessarily feasible for TLD-IP. We next solve TLD-IP, but fixing variables $x = x^1$ and $y = y^1$ to re-position trailers and create a feasible solution to TLD-IP. Note that with these variable fixed, TLD-IP reduces to a minimum cost network flow problem. The results of this experiment are provided in Table 15, where we report the cost savings achieved by our integrated approach ("Integrated") and the sequential approach ("Sequential").

Table 15: Load Plan Savings For Each Week

	Apr08-W1	Apr08-W2	Apr08-W3	Mar09-W1	Mar09-W2	Mar09-W3	Mar09-W4
Integrated	4.49	3.93	4.77	3.73	3.29	3.69	3.77
Sequential	2.49	2.49	2.10	3.21	2.82	3.08	2.90

Clearly, our integrated approach leads to significant increases in savings when compared to an idealized sequential approach.

Table 16: Cost Component Comparison

	Loaded Trailers		Empty Trailers		Handling	
	Integrated	Sequential	Integrated	Sequential	Integrated	Sequential
Apr08-W1	96.74	96.11	78.82	112.64	97.02	97.66
Apr08-W2	97.02	96.09	82.13	111.40	97.81	98.43
Apr08-W3	96.11	96.81	79.80	111.69	98.11	97.06
Mar09-W1	97.69	96.04	79.70	102.28	97.15	97.85
Mar09-W2	98.33	96.17	79.77	106.93	96.86	97.48
Mar09-W3	98.31	96.22	77.25	103.32	96.02	97.16
Mar09-W4	98.46	96.29	75.48	106.63	96.57	96.30

In Table 16, we compare the individual cost components (loaded trailer costs, empty

trailer costs, and handling costs) of the load plans produced by each approach relative to the carrier's load plan. Specifically, we define c_{Init}^{comp} to be the cost for a component in the carrier's load plan and c_{Alg}^{comp} to be the cost for a component of the load plan produced algorithmically, and report in the columns values of $c_{Alg}^{Component}/c_{Init}^{Component} \times 100\%$. While the loaded trailer costs and the handling costs are about the same for both approaches, routing freight and empty trailers sequentially results in an increase in empty trailer costs compared to the carrier's initial load plan.

Finally, we illustrate in Figure 12 the savings found by the neighborhood search for the Apr08-W1 instance over the course of its execution. We see that although we allow the heuristic to run for six hours the vast majority of savings are found in the first three hours, after which the savings found are minimal.

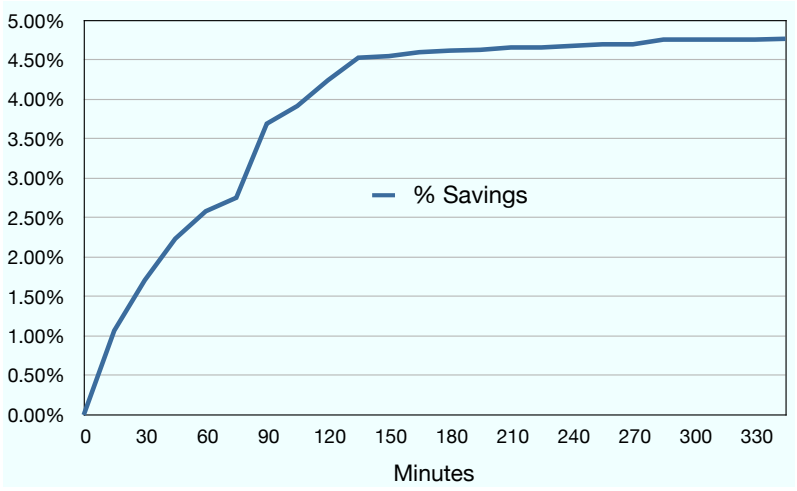

Figure 12: Savings over time

4.7.3 Variations on the Traditional Load Plan

Having established that integer programming based neighborhood search can produce traditional load plans with the potential for significant savings, we next turn to the question of what savings can be achieved by relaxing different constraints of the traditional load plan. Again, savings are computed relative to the cost of the carrier's load plan for the relevant week. Not surprisingly, we see in Table 17 that unrestricted load plans lead to the greatest savings. Note that while day-differentiated load plans lead to significantly greater savings than traditional load plans, same-path load plans do not.

Table 17: Load Plan Savings For Load Plan Variants

		Traditional	Day-Differentiated	Same-Path	Unrestricted
Apr08-W1	$\Delta Cost$	4.49	6.08	4.93	7.41
	ΔPPT	2.17	3.25	2.50	3.47
Apr08-W2	$\Delta Cost$	3.94	6.11	4.90	7.62
	ΔPPT	1.37	2.77	2.20	4.60
Apr08-W3	$\Delta Cost$	4.77	6.93	5.14	8.05
	ΔPPT	3.13	4.77	3.40	5.81
Mar09-W1	$\Delta Cost$	3.73	6.51	4.01	7.60
	ΔPPT	2.37	4.88	2.60	5.78
Mar09-W2	$\Delta Cost$	3.29	6.02	3.44	7.22
	ΔPPT	1.68	3.90	1.91	5.06
Mar09-W3	$\Delta Cost$	3.69	6.51	3.85	7.60
	ΔPPT	1.66	4.24	1.79	5.02
Mar09-W4	$\Delta Cost$	3.77	6.76	3.83	7.89
	ΔPPT	1.27	4.07	1.59	5.18

We next compare the individual cost components of each load plan variant in Table 18 where we use the following abbreviations: Traditional (T), Day-Differentiated (DD), Same-Path (SP), and Unrestricted (U). We see that the opportunity to vary freight routing decisions by day (which is present in both the Day-Differentiated and Unrestricted load plans) allows for a significant decrease in empty trailer repositioning costs.

Table 18: Load Plan Variants Cost Component Comparison

	Loaded Trailers				Empty Trailers				Handling			
	T	DD	SP	U	T	DD	SP	U	T	DD	SP	U
Apr08-W1	96.74	96.26	96.27	94.66	78.82	68.62	78.09	68.20	97.02	93.61	96.98	93.29
Apr08-W2	97.02	95.88	96.07	94.07	82.13	73.35	81.76	71.68	97.81	93.24	96.52	93.45
Apr08-W3	96.11	95.08	96.19	93.64	79.80	70.55	75.80	71.07	98.11	93.16	97.04	93.02
Mar09-W1	97.69	95.55	97.61	94.52	79.70	75.70	78.68	71.02	97.15	91.83	96.25	91.11
Mar09-W2	98.33	96.35	98.23	95.28	79.77	76.51	78.71	69.78	96.86	90.65	96.98	91.58
Mar09-W3	98.31	96.03	98.18	94.97	77.25	72.65	76.12	68.92	96.02	91.44	96.18	91.49
Mar09-W4	98.46	97.99	99.37	96.65	75.48	71.58	75.91	68.68	95.67	93.76	99.05	94.35

In both a Day-Differentiated and Unrestricted load plan, we relax the restriction that freight between two terminals must follow the same sequence of directs regardless of the day of origin. Therefore, we present in Table 19 and 20 how often different paths are used during the week. Specifically, Tables 19 and 20 report what percentage of origin-destination pairs

Table 19: Allowing Different Paths on Different Days-Apr08

# Diff.	Apr08-W1		Apr08-W2		Apr08-W3	
Paths	DD	U	DD	U	DD	U
1	79.57	75.35	80.40	75.76	77.78	74.95
2	17.06	20.31	16.94	19.86	18.47	20.08
3	3.12	3.79	2.43	3.99	3.28	4.24
4	0.25	0.53	0.21	0.34	0.42	0.70
5	0.00	0.02	0.02	0.04	0.04	0.04

Table 20: Allowing Different Paths on Different Days - Mar09

# Diff.	Mar09-W1		Mar09-W2		Mar09-W3		Mar09-W4	
Paths	DD	U	DD	U	DD	U	DD	U
1	75.16	66.11	73.94	65.65	73.37	65.52	71.36	63.33
2	19.69	25.13	20.59	25.68	21.07	26.36	22.09	27.03
3	4.39	7.38	4.85	7.32	4.86	6.86	5.70	7.98
4	0.73	1.30	0.60	1.19	0.65	1.20	0.76	1.57
5	0.04	0.09	0.02	0.16	0.04	0.05	0.09	0.09

whose path could have been changed during the execution of the heuristic sends freight along 1, 2, 3, 4, or 5 paths during the week. For the Day-Differentiated load plan, we see that the majority of origin-destination pairs still use a single path for each weekday. Roughly 19.5% of origin-destination pairs sends freight along two paths during the week. Finally, only few origin-destination pairs send freight on three or more paths. From an implementation perspective, this "path profile" is desirable since the Day-Differentiated load plan does not represent a significant shift from how freight would be routed under a traditional load plan. For the Unrestricted load plan, we see that while slightly fewer origin-destination pairs send freight along a single path, compared to the Day-Differentiated load plan, the vast majority still do.

In contrast, the Same-Path load plan retains the constraint that freight between two terminals must follow the same sequence of directs each day, but relaxes the in-tree structure of a traditional load plan. Specifically, these variants allow two commodities with the same destination to take different outbound directs from a terminal. Day-Differentiated plans also allow this, but only on different weekdays. In Tables 21 and 22, we report how many terminals use different outbound directs at some point during the week to route freight to the same destination. Note for both tables we again only report the origin-destination pairs whose path could have been changed during the execution of the heuristic. We see that relaxing the tree structure while enforcing the same path constraints generates load plans that are very similar to traditional load plans; *i.e.*, they rarely violate the single outbound direct constraint. However, the Day-Differentiated and Unrestricted load plans that we generate are such that the vast majority of the single outbound direct constraints are satisfied. Even when multiple outbound directs are used, rarely are more than two used. This is not completely surprising, since using a single outbound direct for freight destined

for a specific terminal is a form of consolidation.

Table 21: % Terminals That Load a Single Destination on Multiple Directs - Apr08

# Outbound	W1			W2			W3		
Directs	SP	DD	U	SP	DD	U	SP	DD	U
1	97.54	78.84	74.64	97.79	80.06	75.43	98.06	77.38	74.50
2	2.39	17.92	20.71	2.10	17.22	19.72	1.94	19.03	20.01
3	0.00	3.00	3.91	0.11	2.57	4.27	0.00	3.17	4.67
4	0.00	0.21	0.69	0.00	0.13	0.49	0.00	0.38	0.73
5	0.00	0.04	0.02	0.00	0.02	0.06	0.00	0.04	0.08
6+	0.00	0.00	0.02	0.00	0.00	0.02	0.00	0.00	0.00

Table 22: % Terminals That Load a Single Destination on Multiple Directs - Mar09

# Outbound	W1			W2			W3			W4		
Directs	SP	DD	U	SP	DD	U	SP	DD	U	SP	DD	U
1	98.45	75.81	66.69	98.71	74.30	65.61	98.63	73.86	65.48	98.06	77.38	74.50
2	1.55	19.32	24.59	1.25	20.80	25.85	1.34	20.96	26.48	1.94	19.03	20.01
3	0.00	4.22	7.18	0.04	4.41	6.99	0.03	4.50	6.86	0.00	3.17	4.67
4	0.00	0.62	1.32	0.00	0.47	1.39	0.00	0.65	1.07	0.00	0.38	0.73
5	0.00	0.04	0.18	0.00	0.02	0.16	0.00	0.04	0.11	0.00	0.04	0.08
6+	0.00	0.00	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

CHAPTER V

CONCLUSIONS AND FUTURE RESEARCH

This dissertation studied the design of algorithms that use both heuristic and exact optimization techniques and considered their application to network design problems of both academic and real-world interest.

Chapter 2 presented a local search heuristic for a classic discrete optimization problem, the Multi-commodity Fixed Charge Network Flow (MCFCNF) problem. The heuristic, named IP Search, searches exponential-sized neighborhoods using integer programming technology. Searching for improving solutions by solving integer programs enables the heuristic to handle many variants of the problem, e.g., with arc capacities, without arc capacities, commodities have to be routed along a single path, commodities can be routed along multiple paths, etc. To search for improving primal solutions, IP Search solves the compact formulation of MCFCNF with many variables fixed to their value in the best known solution. In addition to producing primal solutions, IP Search produces dual bounds by coupling the extended formulation of MCFCNF with lagrangean techniques. Neither of these tasks require an entire instance to be loaded into memory, making IP Search well-suited to large instances. The computational experiments demonstrate that IP Search is superior to existing metaheuristics and pure integer programming approaches when it comes to producing high-quality primal solutions quickly. However, the dual bound produced by IP Search is relatively weak because the MCFCNF satisfies the integrality property, e.g. the optimal value of the extended and compact formulation linear programming relaxations are the same, and because the linear programming relaxation of the compact formulation is typically very weak.

Chapter 2 presented compelling evidence that one can develop a powerful IP-based search heuristic based on solving restrictions wherein structural knowledge of the problem is used to fix variables. In Chapter 3 we studied how we can abstract the ideas underlying

this heuristic and develop an algorithm for general integer programs. Instead of defining restrictions by fixing variables to values we simply required that constraints are added to the original problem. We then presented an extended formulation that models both the problem and the restriction to solve to produce the optimal solution. We then used column generation both for creating restrictions and producing a dual bound on the problem. We also saw that our extended formulation provided natural methods for producing integer programs that represent a neighborhood of the best known solution. Finally, while the approach is not completely generic, we note that applying it to a specific problem class requires very few decisions. Computational experiments demonstrated that when applied to the MCFCNF, the approach is able to produce both high quality primal solutions and strong dual bounds quickly.

Two important properties of the heuristic presented in Chapter 2 served as motivation for the work presented in Chapter 4. First, the heuristic did not require an entire instance to be loaded into memory, suggesting it can be applied to very large problem instances. Second, it could easily handle different problem variants, suggesting it can accommodate real-world problems that have flexible definitions.

Chapter 4 first presented a large scale dynamic model for the service network (or *load plan*) design problem for Less-Than-Truckload freight transportation carriers. We extended existing models of carrier operations by routing loaded and empty trailers simultaneously, measuring time at a level of detail appropriate for a carrier with highly time-sensitive shipments and recognizing consolidation opportunities in time. For a national carrier, these latter two model characteristics yield very large instances. Thus, using the insights from Chapter 2 and our understanding of the structure of a load plan, we presented a local search heuristic wherein searching a neighborhood equates to re-optimizing all freight paths into a single destination terminal, while keeping all other freight paths fixed. Since an integer program is solved to re-optimize freight paths, we presented valid inequalities and preprocessing techniques that speed up its solve time. The computational experiments demonstrated that the heuristic can produce load plans with the potential for significant cost savings for a national carrier. In addition, discussions with a national carrier have lead us to believe that

the simultaneous routing of loaded and empty trailers is critical to ensuring that the model produces load plans a carrier would execute. Next, Chapter 4 studied the savings potential of relaxing some self-imposed rules that constrain how freight traditionally flows through an LTL network. In particular, a traditional load plan assumes that the same freight routing decisions are made every day. We studied the savings potential of allowing freight routing decisions to vary by day by using the same heuristic albeit with a slight redefinition of the integer program that re-optimizes freight paths. The computational experiments demonstrated that while these *day-differentiated* load plans have great potential for cost savings, most freight routing decisions do not vary by day. Hence, we feel that the day-differentiated load plans produced by the heuristic are close enough in structure to traditional load plans that a carrier would find them implementable.

Each of the three chapters presented interesting topics for future research. The heuristic presented in Chapter 2 made extensive use of both the compact and extended formulation of MCFCNF and we believe this is one reason for its success. In fact, many of the ideas that were presented in the context of MCFCNF could be applied to other hard discrete optimization problems. It will be especially interesting to apply these ideas to a problem such as the Generalized Assignment Problem wherein the extended formulation provides a much tighter dual bound than the compact formulation. This tighter bound requires a more difficult pricing problem than the pricing problem for MCFCNF; thus it will be interesting to see how robust IP Search is to the difficulty in solving the pricing problem. This trade-off also suggests there may be advantages to parallelizing the heuristic.

One of our goals for the work in Chapter 3 was to develop an approach that could be easily applied to new problem classes. In addition, we wanted to explore how to develop an approach that leveraged the ability to perform tasks in parallel. While the computational results when applying the approach to MCFCNF are impressive, applying the approach to another problem will be a good test of whether we met those goals. It will be especially interesting to study whether we can turn the approach into a true "black-box" by automating the choice of restriction structure.

We see three streams of research emanating from the work in Chapter 4. The first is to

improve our models of LTL operations. Although we have modeled LTL operations much more accurately than current approaches, when we route trailers we still ignore drivers and the constraints governing what they can do. It will be interesting to see if we can incorporate the building of driver tours to cover trailer routes into this approach and still produce savings for realistically sized instances. The second stream is to investigate extending the approach to include facility capability decisions, namely which existing terminals should support the transfer of freight. The third is to study the potential of real-time load planning. By deploying wireless devices in their pickup and delivery operations carriers have tremendous visibility of the freight that is entering the linehaul network on a given day. Hence, while load plan design is typically thought of as a planning problem and is based on projected freight volumes, one can easily envision the savings potential of designing a load plan to be executed that night based on the improved information regarding that day's freight volume information.

REFERENCES

[1] Aarts, E. and Lenstra, J. K., eds., *Local Search in Combinatorial Optimization*. Princeton University Press, 2003.

[2] Ahuja, R., Ergun, O., Orlin, J., and Punnen, A., "A survery of very large scale neighborhood search techniques," *Discrete Applied Mathematics*, vol. 123, pp. 75–102, 2002.

[3] Archetti, C., Speranza, M. G., and Savelsbergh, M. W. P., "An optimization-based heuristic for the split delivery vehicle routing problem," *Transportation Science*, vol. 42, pp. 22–31, 2008.

[4] Balinski, M., "Fixed cost transportation problem," *Nav. Res. Logistics Q.*, vol. 8, pp. 41–54, 1961.

[5] Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., and Vance, P. H., "Branch-and-price: Column generation for solving huge integer programs," *Operations Research*, vol. 46, pp. 316–329, 1996.

[6] Blum, C. and Roli, A., "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Computing Surveys*, vol. 35, pp. 268–308, 2003.

[7] Crainic, T., Frangioni, A., and Gendron, B., "Bundle-based relaxation methods for multicommodity capacitated fixed charge network design," *Discrete Applied Mathematics*, vol. 112, pp. 73–99, 2001.

[8] Crainic, T. and Gendreau, M., "Cooperative parallel tabu search for capacitated network design," *Journal of Heuristics*, vol. 8, pp. 601–627, 2002.

[9] Crainic, T., "Service network design in freight transportation," *European Journal of Operations Research*, vol. 122, pp. 272–288, 2000.

[10] Crainic, T., Gendreau, M., and Farvolden, J., "A simplex-based tabu search method for capacitated network design," *INFORMS J. Comput.*, vol. 12, pp. 223–236, 2000.

[11] Crainic, T., Gendron, B., and Hernu, G., "A slope scaling/lagrangean perturbation heuristic with long-term memory for multicommodity capacitated fixed-charge network design," *Journal of Heuristics*, vol. 10, pp. 525–545, 2004.

[12] Danna, E., Rothberg, E., and Le Pape, C., "Exploring relaxation induced neighborhoods to improve MIP solutions.," *Mathematical Programming*, vol. 102, pp. 71–90, 2005.

[13] De Franceschi, R., Fischetti, M., and Toth, P., "A new ILP-based refinement heuristic for vehicle routing problems," *Mathematical Programming B*, vol. 105, pp. 471–499, 2006.

[14] Fischetti, M. and Lodi, A., "Local branching," *Mathematical Programming*, vol. 98, pp. 23–47, 2003.

[15] Ghamlouch, I., Crainic, T., and Gendreau, M., "Cycle-based neighborhoods for fixed charge capacitated multicommodity network design," *Operations Research*, vol. 51, pp. 655–667, 2003.

[16] Ghamlouch, I., Crainic, T., and Gendreau, M., "Path relinking, cycle-based neighborhoods and capacitated multicommodity network design," *Annals of Operations Research*, vol. 131, pp. 109–133, 2004.

[17] Gropp, W., *Using MPI: Portable Parallel Programming with the Message Passing Interface*. The MIT Press, 1999.

[18] Gu, Z., Nemhauser, G. L., and Savelsbergh, M. W., "Lifted cover inequalities for 0-1 integer programs: Computation," *INFORMS J. Comput.*, vol. 10, pp. 427–437, 1998.

[19] Holmberg, K. and Yuan, D., "A lagrangean heuristic based branch-and-bound approach for the capacitated network design problem," *Operations Research*, vol. 48, pp. 461–481, 2000.

[20] ILOG, *ILOG CPLEX User's Manual*. ILOG, 2008.

[21] Jarrah, A., Johnson, E., and Neubert, L., "Large-scale, less-than-truckload service network design," *Operations Research*, vol. 57, pp. 609–625, 2009.

[22] Kim, D. and Pardalos, P., "A solution to the fixed charge network flow problem using a dynamic slope scaling procedure," *Operations Research Letters*, vol. 24, pp. 195–203, 1999.

[23] Mladenovi, N. and Hansen, P., "Variable neighborhood search," *Computers and Operations Research*, vol. 24, pp. 1097–1100, 1997.

[24] Nemhauser, G. L. and Wolsey, L. A., *Integer and Combinatorial Optimization*. Wiley, New York, 1988.

[25] Pederson, M. B., Crainic, T. G., and Madsen, O. B., "Models and tabu search metaheuristics for service network design with asset-balance requirements," *Transportation Science*, vol. 43, 2009.

[26] Pessoa, A., de Aragao, M. P., and Rodrigues, R., "Algorithms over arc-time indexed formulations for single and parallel machine scheduling problems," *Optimization Online*, 2008.

[27] Pisinger, D. and Ropke, S., "A general heuristic for vehicle routing problems," *Computers and Operations Research*, vol. 34, pp. 2403–2435, 2007.

[28] Powell, W. B., "A local improvement heuristic for the design of less-than-truckload motor carrier networks," *Transportation Science*, vol. 20, pp. 246–257, 1986.

[29] Powell, W. B. and Koskosidis, I. A., "Shipment routing algorithms with tree constraints," *Transportation Science*, vol. 26, pp. 230–245, 1992.

[30] Powell, W. B. and Sheffi, Y., "The load planning problem of motor carriers: Problem description and proposed solution approach," *Transportation Research–A*, vol. 17A, pp. 471–480, 1983.

[31] Powell, W. B. and Sheffi, Y., "Design and Implementation of an Interactive Optimization System for Network Design in the Motor Carrier Industry," *Operations Research*, vol. 37, pp. 12–29, 1989.

[32] Powell, W. and Farvolden, J., "Subgradient methods for the service network design problem," *Transportation Science*, vol. 28, pp. 256–272, 1994.

[33] Powell, W. and Sheffi, Y., "Design and implementation of an interactive optimization system for network design in the motor carrier industry," *Operations Research*, vol. 37, pp. 12–29, 1989.

[34] Ropke, S. and Pisinger, D., "An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows," *Transportation Science*, vol. 40, pp. 455–472, 2006.

[35] Savelsbergh, M. and Song, J.-H., "An optimization algorithm for inventory routing with continuous moves," *Computers and Operations Research*, vol. 35, pp. 2266–2282, 2008.

[36] Schmid, V., Doerner, K. F., Hartl, R. F., Savelsbergh, M. W. P., and Stoecher, W., "A hybrid solution approach for ready-mixed concrete delivery," *Transportation Science*, vol. 43, pp. 70–85, 2008.

[37] Wieberneit, N., "Service network design for freight transportation: a review," *OR Spectrum*, vol. 30, 2008.